AFHRL-TR-89-42

(2)

AIR FORCE HUMAN RESOURCES LABORATORY

SHAPE DISCRIMINATION RESEARCH USING AN IBM PC

Christopher D. Voltz
George A. Geri

University of Dayton Research Institute
300 College Park Avenue
Dayton, Ohio 45469

OPERATIONS TRAINING DIVISION
Williams Air Force Base, Arizona 85240-6457

March 1990
Final Technical Report for Period October 1987 - July 1989

AIR FORCE SYSTEMS COMMAND
BROOKS AIR FORCE BASE, TEXAS 78235-5601

NOTICE

When Government drawings, specifications, or other data are used for any purpose other than in connection with a definitely Government-related procurement, the United States Government incurs no responsibility or any obligation whatsoever. The fact that the Government may have formulated or in any way supplied the said drawings, specifications, or other data, is not to be regarded by implication, or otherwise in any manner construed, as licensing the holder, or any other person or corporation; or as conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.

The Public Affairs Office has reviewed this report, and it is releasable to the National Technical Information Service, where it will be available to the general public, including foreign nationals.

This report has been reviewed and is approved for publication.

PAUL M. CHOUDEK, Capt, USAF
Contract Monitor

DEE H. ANDREWS, Technical Director
Operations Training Division

HAROLD G. JENSEN, Colonel, USAF
Commander

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	March 1990	Final - October 1987 - July 1989

4. TITLE AND SUBTITLE
Shape Discrimination Research Using an IBM PC

5. FUNDING NUMBERS
C - F33615-87-C-0012
PE - 61102F, 62205F
PR - 2313, 1123
TA - T3, 03
WU - 12, 83

6. AUTHOR(S)
Christopher D. Voltz
George A. Geri

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
University of Dayton Research Institute
300 College Park Avenue
Dayton, Ohio 45469

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Operations Training Division
Air Force Human Resources Laboratory
Williams Air Force Base, Arizona 85240-6457

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
AFHRL-TR-89-42

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

A menu-driven program is described that is designed to control the presentation of visual stimuli and to collect and analyze response data in shape discrimination research. The program presents high resolution stimuli on an IBM PC equipped with an IBM Professional Graphics Adaptor (PGA) and an analog monitor. Bit-plane layering is employed to allow multiple test and adapting stimuli (1- to 2-bit gray scale) to be presented with specifiable interstimulus intervals. The program implements a double-random staircase paradigm, collects a subject's responses via the computer's parallel port, and analyzes the resulting data. In addition, auxiliary programs are provided for generating Fourier Descriptor shape stimuli, for modifying stimulus parameters, and for formatting and storing the final stimuli for use by the main program.

14. SUBJECT TERMS
display hardware
laboratory computer software
visual research

15. NUMBER OF PAGES
82

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

SUMMARY

A menu-driven program is described that is designed to generate and present visual stimuli for use in shape discrimination research. The main function of the program is to present stimuli on an IBM PC equipped with a Professional Graphics Adapter (PGA). The program is modular in design and may be easily modified for use with other laboratory computers. Provision has also been made for the presentation of several adapting and/or noise (masking) stimuli and for accurately controlling the temporal relationship of each to the primary stimuli. In addition, the program initializes the PGA, accepts subject specifications, and collects response data via a switch interfaced to the computer's parallel port. The response data are collected using a procedure that changes the test stimuli depending on the subject's previous response. Program options allow both analysis and plotting of the response data. Auxiliary programs are provided for generating stimulus outlines of various shapes; for modifying stimulus parameters such as spatial location, size, contour amplitude, and phase; and for formatting and storing the final stimuli such that they can be used by the main program.

PREFACE

This research was performed in support of the Training Technology Planning Objective of the Research and Technology Plan at the Operations Training Division of the Air Force Human Resources Laboratory, Williams Air Force Base, Arizona. The general objective of this training research and development program is to identify and demonstrate cost-effective strategies and new training systems for developing and maintaining combat effectiveness. The purpose of the present experiment was to elucidate the basic mechanisms underlying visually guided behavior in flight simulators.

The authors thank Susan Baroff, who provided the auxiliary program for generating the Fourier Descriptor stimuli. Drs. Don Lyon, Yehoshua Zeevi, and John Uhlarik contributed to the design of the experimental procedures which have been implemented in the programs described in this report. We also thank Dr. Elizabeth Martin for her support and encouragement. This research was supported by the Air Force Office of Scientific Research, Work Unit 2313-T3-12, Cognitive Aspects of Flight Training, Dr. Elizabeth Martin, Principal Investigator; and by Air Force Contract F33615-87-C-0012 (UDRI), Flying Training Research Support, Capt. Paul M. Choudek, Contract Monitor.

TABLE OF CONTENTS

SHAPE DISCRIMINATION RESEARCH USING AN IBM PC

I. INTRODUCTION

The program FOURIER4.BAS, which is described here, has been used to conduct shape discrimination research in the Basic Research Laboratory at the Air Force Human Resources Laboratory, Williams AFB, Arizona. Shape stimuli, known as Fourier Descriptors (Kuhl & Giardina, 1982; Zahn & Roskies, 1972), were generated off-line and subsequently presented to observers who were asked to distinguish the various higher harmonic stimuli from a fundamental (elliptical) stimulus. The stimuli were presented in pairs whose members were displayed at various distances to the left and right of the fixation point-- thus constituting a two-alternative, spatial forced-choice paradigm. The amplitude associated with each harmonic stimulus was either reduced or increased on each trial depending on whether or not the observer was able to distinguish the two stimuli on the previous trial. The resulting data were analyzed using standard techniques and plotted when required.

As the program FOURIER4.BAS has been used as part of an ongoing visual shape discrimination research project, portions of the program and its documentation are specific to the stimuli and techniques used in that research. The program is, however, modular in design and we have indicated in the documentation some of the changes which may be made to adapt it to other experimental paradigms. For instance, stimuli other than Fourier descriptors may be used provided certain conventions are observed (e.g., the stimulus file must be in the PGA Run Length Encoded format [IBM, 1984], and the stimulus file names must be of the particular format used here). Also, changes can be made in the psychophysical procedures used to collect data as well as in stimulus characteristics such as color and intensity, the time interval between adapting and test stimuli, and the tradeoff between the number of stimuli and the number of gray-levels in each stimulus. Familiarity with the program's structure and function may be required in order to make some of the changes mentioned; and in some cases, readers may find it more expedient to use individual modules in their own programs.

II. DOCUMENTATION FOR PROGRAM "FOURIER4.BAS"

FOURIER4.BAS is a menu-controlled program whose primary function is to present visual stimuli on an IBM PC equipped with an IBM PGA. The program also (a) initializes the display controller, (b) randomizes trial blocks, (c) accepts subject specifications, (d) collects response data, (e) analyzes response data, (f) plots mean data, (g) demonstrates the chosen stimulus sequence, and (h) tests the subject response box. Note that the PGA requires an analog monitor (an IBM 8514 monitor, any NEC Multisync monitor, etc.). The program will present any images previously created using PGAGRAPH.BAS. (If single images are desired, a modification must be made in module BEGIN.TESTING wherever there are two consecutive calls to LOAD.SCREEN). The images to be used must reside in files named using the following convention: WWWXXYZ.IMG where "WWW" is the image specification (e.g., "P40" denotes a peripherally presented 40-mm stimulus), "XX" is the number of the harmonic (or other stimulus parameter), "Y" is the step level (0-9, and corresponding in the FD study to the harmonic amplitude), and "Z" is either 'L' or 'R' indicating whether the image is to be displayed to the left or to the right of the fixation point (assuming that image pairs are required). The field "WWW" may consist of any alphanumeric characters, whereas the fields "XX" and "Y" are numeric and padded as necessary with leading zeroes. Because FOURIER4.BAS, as presently configured, presents pairs of images, the images created using the program PGAGRAPH should, in general, not overlap (see description of the area-of-interest [AOI] box in the PGAGRAPH documentation). Further, if the fixation point is used, the images should not overlap the fixation point image.

The menu-option "ENTER STIMULUS INFO" is usually run once at the beginning of each experiment. This option prompts the user to enter the number of stimulus blocks (i.e, harmonics or other stimulus parameter), the particular harmonic levels to be run, and the starting amplitude (specified as a step from 0 to 9) of the upper and lower staircases. This information is stored in the file Fourier4.dat. The menu-options "INITIALIZE PGA," "RANDOMIZE TRIAL DATA," "ENTER SUBJECT DATA," and "COLLECT DATA" are run prior to each experimental session. For each of the stimulus blocks specified above, a double-random staircase sequence is implemented until the chosen number of response reversals (%max.num.trials) has occurred. In each trial, the subject is shown a pair of stimuli, one of which is a standard (here the first harmonic ellipse) that is presented randomly to the left or right of fixation. The second member of the pair is the stimulus associated with the block presently selected and the particular staircase step being run. The amplitude corresponding to each staircase step is determined by the specifications chosen when the images were created (using PGAGRAPH.BAS in the case of the FD stimuli). Following stimulus presentation, the subject responds by depressing a debounced switch corresponding to that side of the screen which appeared to contain the test stimulus. The switch is connected to parallel port #1 such that bits 7 and 3 (bits numbered left to right from 7 to 0) are toggled at each keypress. For the data analysis

and plotting output to be valid, it is necessary that the amplitude increment, chosen during image generation, matches the variable "amp.step" in section "GLOBAL VARIABLE INITIALIZATION." Note that data may be collected using unequal staircase step sizes, but in that case the data analysis cannot be done by this program.

To collect data, first initialize the display controller, randomize the stimulus blocks, and enter the subject information using the appropriate menu options. Next, select "DATA COLLECTION" and respond to all prompts (see also above). There are then two run options corresponding to whether or not an adapting stimulus is specified in menu-option "DATA COLLECTION."

If an adapting stimulus is specified, the following sequence is followed:

(1) noise screen displayed until either switch is pressed,
(2) stimulus in current block is previewed for subject,
(3) noise screen displayed until next switch press,
(4) adapting stimulus is displayed for 2 sec,
(5) noise screen is displayed for 1 sec,
(6) stimulus pair is displayed for 150 msec,
(7) noise screen is displayed,
(8) subject's response is recorded,
(9) return to (1) if not end of block.

If no adapting stimulus is specified, the following sequence is followed:

(1) display noise screen for 1 sec,
(2) 1000 ms later, display the stimulus for 150 msec,
(3) the noise screen is displayed,
(4) the subject's response is recorded,
(5) return to (1) if not end of block.

The module labeled BEGIN.TESTING is used to display the chosen stimuli and collect the experimental data. As it is the most complex of the modules, we provide the following description of its function.

The module does the following at the beginning of each experimental session (assuming that an adapting stimulus has been chosen):

(A) opens the random number data file (Fourier4.dat),

(B) reads the total number of blocks and the line number of the stimulus to be displayed next,

(C) moves the pointer to that line (usually representing the first block unless this is a continuation of a previous session).

The module does the following [through (P)] for each of the stimulus blocks (whose number is specified by "num.block%"):

(D) reads the upper and lower staircase starting points, reads the number of the stimulus to be displayed next (2, 4, 6, etc. in the case of the FD stimuli), and reseeds the random number generator.

(E) initializes variables to be used for current block,

(F) switches to the PGA display mode, a displays blank screen, loads (but does not display) the chosen noise screen and displays the fixation point, loads the preview stimulus, and beeps,

(G) waits for a signal from the response box,

(H) displays the preview stimulus, displays a blank screen, waits for a signal from the response box, and displays the noise screen.

The module does the following [through (O)] until the maximum number of reversals (%max.num.trials) has occurred:

(I) if the number of times a given staircase has been used (num.case%) is greater than the maximum (max.num.case%), then switches to the other staircase, or else picks a staircase at random;

(J) determines the next amplitude to use for the current staircase and randomly chooses the side of the screen (side%=1 => left, side%=2 => right) on which to display the comparison (here zero-harmonic) stimulus;

(K) loads in the test and comparison stimuli, presents the adapting stimuli if the adapting flag (adapt.flag%) is set, and then presents the noise screen, the test and comparison stimuli, and the noise screen again;

(L) gets the subject's response, and writes to the output data file (opened by the "enter.subject.data" module) the current stimulus number, the current amplitude, which staircase (stair.case%=0 => upper, =1 => lower) was used, whether the response was correct (1) or incorrect (0), whether the subject responded left (0) or right (1), and the side on which the comparison stimulus was presented;

(M) determines the amplitude to display when the current staircase is used next (as determined by the amplitude level [0-9] stored in "up.case%" and "low.case%"), sets the reversal flag (reversed%) if a reversal has occurred, and, if a reversal has occurred, increments the count of the number of reversals (num.trials%);

(N) if the subject responded incorrectly, then sounds the error tone,

(O) if the escape key has been pressed, then switches to the CGA display mode, closes the output file and returns to the main menu, otherwise returns to I);

(P) prints a blank line to the output file, increments the block number, and returns to D);

(Q) switches to the CGA display mode, prompts user for comments which are appended to the end of the output file, and closes the input and output files.

III. LISTING OF PROGRAM "FOURIER4.BAS"

```
1:  '              FILENAME:       fourier4.bas
2:  '              CREATED:        6/08/87
3:  '              PROGRAMMER:     Christopher Voltz - UDRI
4:  '              LAST MODIFIED:  5/17/89
5:  '              TARGET:         IBM PC w/PGA
6:  '              LANGUAGE:       Turbo BASIC v. 1.00
7:  '
8:  '
9:  '                              PROGRAM PURPOSE
10: '
11: '      This is a menu-controlled program whose primary function is to present
12: '    pairs of stimuli on an IBM PC equipped with a Professional Graphics Adapter
13: '    (PGA).  See FOURIER.DOC for detailed documentation.
14:
15:
16:
17: '
18: '  ┌──────────────────────────────────────────────────────────┐
19: '  │                    CONSTANT DEFINITIONS                  │
20:           └──────────────────────────────────────────────────────────┘
21:                  ' *** system specific constants ***
22:           %buffer.size  = 15000    'number of bytes to read in per disk access
23:                                    '(increase for slow disks; not to exceed 32767)
24:           %escape.key   = 27       'scan code returned by escape key
25:
26:                  ' *** menu display constants ***
27:           %entry.indent = 5        'number of spaces to indent data entry prompts
28:           %error.color  = 28       'blinking light red
29:           %menu.color   = 15       'bright white
30:           %menu.indent  = 10       'number of spaces to indent menus
31:           %option.color = 3        'cyan
32:
33:                  ' *** response box constants ***
34:           %bit.mask         = &B10001000  'mask to clear extra bits
35:           %right.button     = &B10000000  'mask for second button
36:           %different.button = &B10000000
37:           %left.button      = &B00001000  'mask for first button
38:           %same.button      = &B00001000
39:           %switch.port      = &H379       'I/O address of switch port (LPT1:)
40:
41:                  ' *** boolean definitions ***
42:           %false = 0       'recognized by the compiler as false
43:           %true  = -1      'recognized by the compiler as true
44:
45:                  ' *** staircase constants ***
46:           %num.noise      =  5    'number of noise screens (1-9 max)
47:           %max.num.trials = 30    'number of reversals
48:           %max.amplitude  =  9    'maximum possible amplitude
49:           %max.num.case   =  2    'maximum number of repeats of a case
50:           %min.amplitude  =  0    'minimum possible amplitude
51:
52:                  ' *** constants required to analyze data
53:           %ati            =  5    'analyze table indention (in spaces)
54:           %header.size    = 12    'number of lines to read to skip header
```

```
55:         %num.harmonics  =  6     'number of harmonics in a file
57:
58:
59: '
60: '  ┌──────────────────────────────────────────────────────────┐
61: '  │                   INCLUDE DECLARATIONS                   │
62:
63:         $INCLUDE "PGA.inc"              'PGA communication routines
64:         $INCLUDE "response.inc"         'response box routines
65:         $INCLUDE "toolbox.inc"          'commonly used routines
66:
67:
68:
69:
70: '  ┌──────────────────────────────────────────────────────────┐
71: '  │                GLOBAL VARIABLE INITIALIZATIONS           │
72: '  └──────────────────────────────────────────────────────────┘
73:
74:                 ' operational flags -- used for error checking primarily
75:         adapt.flag% = %false    'assume adapting stimulus will not be used
76:         data.flag%  = %false    'subject data not loaded yet
77:         demo.flag%  = %false    'if in demonstration mode
78:
79:                 ' filename prefixes -- suffixes are assumed
80:         adapt.file.name$  = "screens\"           'file name of adapting harmonic screen (.IMG)
81:         data.file.name$   = "data\"              'file name of data output
82:         fix.file.name$    = "screens\fixate.img" 'file name of fixation screen
83:         image.file.name$  = "screens\"           'file name of screen images (.IMG)
84:         noise.file.name$  = "screens\noise"      'file name of noise screen (.SCR)
85:         random.file.name$ = "fourier4.dat"       'file name of random data
86:         sample.image$     = "7"                  'file name prefix of sample harmonic
87:         temp.file.name$   = "temp.dat"           'file name of temporary data
88:
89:                 ' recognized keys
90:         escape$           = CHR$(27)           'character string returned by escape key
91:
92:                 ' constants required to analyze data
93:         amp.step! = 0.07                       'amplitude step size
94:
95:
96:
97: '  ┌──────────────────────────────────────────────────────────┐
98: '  │                       MAIN MODULE                        │
99: '  └──────────────────────────────────────────────────────────┘
100:
101:        ' install the error handling system
102:    ON ERROR GOTO error.handler
103:
104:        ' initialize the text screen
105:    SCREEN 0
106:    WIDTH 80
107:    KEY OFF
108:
109: restart:                               ' fatal errors return here
```

```
110:     DO
112:         CALL display.menu(response$)
113:     LOOP UNTIL UCASE$(response$)="X"    'repeat until terminate command given
114:     END
115:
116:
117: error.handler:
118:     CALL PGA.transmit("CA DI,1 ")       ' return to emulator screen
119:     IF (ERR<>55) AND (ERR<>255) THEN    ' testing interrupt or file already open ?
120:          CALL print.error("ERROR "+STR$(ERR)+": "+FN get.error.message$+" AT ADDRESS "+ STR$(ERADR)+".")
121:          RESUME restart                 ' nope, alert user and continue.
122:    ELSE
123:          CLOSE
124:          RESUME restart                 ' yes, restart
125:     END IF
126:
127:
128:
129: '
130: '   SUBROUTINES
131: '
132:
133:
134: '
135: '   ANALYZE.DATA
136: '
137: '
138: '        This module reads in a specified data file and analyzes it
139: '     to produce the estimate of the mean and the standard deviation.
140: '     Refer to "Introduction to Statistical Analysis" by Dixon and
141: '     Massey, McGraw-Hill, 1957 for more information regarding staircase
142: '     calculations.
143: '
144: '
145:
146: SUB analyze.data
147:
148:          '**** GLOBAL VARIABLE DECLARATIONS
149:     SHARED  amp.step!                          'amplitude step size
150:     SHARED  data.file.name$                    'prefix for data files
151:
152:          '**** LOCAL VARIABLE DECLARATIONS
153:     LOCAL  amplitude%                          'number of amplitude => %min.amplitude..%max.amplitude
154:     LOCAL  done%                               'used to determine end of harmonic
155:     LOCAL  est.deviation!                      'estimate of the standard deviation
156:     LOCAL  est.mean!                           'estimate of the mean
157:     LOCAL  file.name$                          'filename of output file
158:     LOCAL  file.names$                         'names of files to analyze
159:     LOCAL  file.only%                          'whether data is only sent to file
160:     LOCAL  harmonic%                           'used to keep track of number of harmonics to be analyzed
161:     LOCAL  index%                              'general loop variable
162:     LOCAL  index.2%                            'general loop variable
163:     LOCAL  low.last%                           'last response on lower staircase
164:     LOCAL  num.files%                          'number of files to analyze
```

```
165:   LOCAL  print.scrn%                              'whether to print the data or not
167:   LOCAL  response.1%                              'response to use for upper staircase
168:   LOCAL  response.2%                              'response to use for lower staircase
169:   LOCAL  reversed.low%                            'lower staircase reversed yet ?
170:   LOCAL  reversed.up%                             'upper staircase reversed yet ?
171:   LOCAL  staircase%                               'which staircase is being used (see note below)
172:   LOCAL  sy2!                                     'Sy^2 used to estimate standard deviation
173:   LOCAL  temp$                                    'used to read in data
174:   LOCAL  up.last%                                 'last response on upper staircase
175:   LOCAL  yn.total!                                'sum of Yi * Ni for all amplitudes
176:   LOCAL  y2n.total!                               'sum of Yi^2 * Ni for all amplitudes
177:
178:   LOCAL  harmonics%()                             'harmonic numbers used
179:   LOCAL  frequency%()                             'frequency of each response at each amplitude for each harmonic
180:   LOCAL  n.total%()                               'total number of
181:   LOCAL  num.responses%()                         'total number of responses
182:   LOCAL  right%()                                 'number of right responses
183:   LOCAL  table!()
184:
185:   DIM harmonics%(%num.harmonics)
186:   DIM frequency%(%max.amplitude, 1, %num.harmonics)
187:   DIM n.total%(1, %num.harmonics)
188:   DIM num.responses%(%num.harmonics)
189:   DIM right%(%num.harmonics)
190:   DIM table!(%max.amplitude, 1, %num.harmonics)
191:
192:   CLS
193:   COLOR %menu.color
194:   PRINT TAB(25);"FOURIER 4: ANALYZE DATA SCREEN"
195:   PRINT
196:   PRINT
197:   COLOR %option.color
198:
199:   PRINT TAB(%entry.indent); : INPUT "ENTER FILENAME OF OUTPUT FILE (.OTS) ";file.name$
200:   PRINT TAB(%entry.indent); : INPUT "NAME OF FILE(S) TO ANALYZE (separated by one space) ";file.names$
201:   file.names$ = file.names$ + " "
202:   num.files% = 0
203:   temp$ = file.names$
204:   WHILE (temp$<>"")
205:       INCR num.files%
206:       temp$ = MID$(temp$, INSTR(temp$, " ")+1)
207:   WEND
208:
209:   file.only% = %TRUE                              'assume output only goes to file
210:   PRINT TAB(%entry.indent);"Display results on screen (Y/<N>) ";
211:   CALL get.key(temp$)
212:   IF UCASE$(temp$) = "Y" THEN file.only% = %FALSE
213:
214:   print.scrn% = %FALSE                            'assume output is not printed
215:   PRINT TAB(%entry.indent);"Print results on printer (Y/<N>) ";
216:   CALL get.key(temp$)
217:   IF UCASE$(temp$) = "Y" THEN print.scrn% = %TRUE
218:
219:   OPEN data.file.name$ + file.name$  + ".OTS" FOR OUTPUT AS #2
```

```
220:   PRINT #2, "FILENAME: ";data.file.name$ + file.name$ + ".OTS",,
222:   PRINT #2, "CREATED: ";date$;" ";time$
223:   PRINT #2, "FILE(S) BEING ANALYZED: ";
224:   temp$ = file.names$
225:   WHILE (temp$<>"")
226:       PRINT #2, LEFT$(temp$, INSTR(temp$, " ")-1) + ".DAT  ";
227:       temp$ = MID$(temp$, INSTR(temp$, " ")+1)
228:   WEND
229:   PRINT #2,
230:   PRINT #2,
231:   PRINT #2, " HARMONIC | .5-MEAN | SD/(n^.5) |  MEAN   |    n    | % Right"
232:
233:   FOR index.2% = 1 TO num.files%
234:
235:     OPEN data.file.name$ + LEFT$(file.names$, INSTR(file.names$, " ")-1) + ".DAT" FOR INPUT AS #1
236:     FOR index% = 1 to %header.size                              'skip header information
237:         INPUT #1, temp$
238:     NEXT
239:     INPUT #1, temp$
240:     done% = (temp$="")                                          'next harmonic?
241:     WHILE (NOT done%) AND (NOT EOF(1))
242:         low.last% = -1                                          'last response on lower staircase
243:         up.last%  = -1                                          'last response on upper staircase
244:         reversed.low% = %FALSE                                  'reset reversal flag
245:         reversed.up%  = %FALSE                                  'reset reversal flag
246:         WHILE (NOT done%) AND (NOT EOF(1))
247:             harmonic% = VAL(temp$)                              'save harmonic number
248:             done% = -1
249:             FOR index% = 0 TO %num.harmonics                    'find index of given harmonic
250:                 IF (harmonic% = harmonics%(index%)) THEN
251:                     done% = index%
252:                 END IF
253:             NEXT
254:             IF (done% = -1) THEN                                'if not found in array
255:                 FOR index% = %num.harmonics TO 0 STEP -1        'find an empty space
256:                     IF (harmonics%(index%)=0) THEN
257:                         done% = index%
258:                     END IF
259:                 NEXT
260:                 harmonics%(done%) = harmonic%                   'save new harmonic #
261:             END IF
262:             harmonic% = done%                                   'set harmonic # to equivalent index
263:             temp$ = MID$(temp$, 3)                              'skip harmonic number
264:             amplitude% = VAL(temp$)                             'save amplitude step number
265:             temp$ = MID$(temp$, 3)                              'skip amplitude info
266:             staircase% = VAL(temp$)                             'save staircase
267:             temp$ = MID$(temp$, 3)                              'skip staircase info
268:             index% = VAL(temp$)                                 'get response code
269:             temp$ = MID$(temp$, 3)                              'skip response code
270:             IF (staircase%=0) THEN                              'check for reversal
271:                 IF (NOT reversed.up%) AND (index%<>up.last%) AND (up.last%<>-1) THEN
272:                     reversed.up% = %TRUE                        'set reversal flag
273:                 END IF
274:                 up.last% = index%                               'save response for next time
```

```
275:            ELSE
277:    `           IF (NOT reversed.low%) AND (index%<>low.last%) AND (low.last%<>-1) THEN
278:                    reversed.low% = %TRUE                          'set reversal flag
279:                END IF
280:                low.last% = index%                                 'save response for next time
281:            END IF
282:            IF ((((staircase%=0) AND reversed.up%) OR ((staircase%=1) AND reversed.low%))) AND _
283:               NOT ((index%=1) AND (amplitude%=0)) THEN
284:                INCR frequency%(amplitude%,index%,harmonic%)       'increment count of response
285:            END IF
286:            INCR num.responses%(harmonic%)                         'increment number of responses
287:            IF (VAL(temp$)=1) THEN                                 'if subject pressed "right"
288:                INCR right%(harmonic%)
289:            END IF
290:            INPUT #1, temp$
291:            done% = (temp$="")
292:        WEND
293:
294:       INPUT #1, temp$
295:       done% = (temp$="")
296:    WEND
297:
298:    IF (num.files%<>1) THEN
299:        file.names$ = MID$(file.names$, INSTR(file.names$, " ")+1)
300:    END IF
301:    CLOSE #1
302:
303:  NEXT                                                             'next file
304:
305:
306:                '*** calculate Ni totals for each type of response at each harmonic
307:  FOR index% = 0 TO 1
308:        FOR amplitude% = 0 TO %max.amplitude
309:            FOR harmonic% = 0 TO %num.harmonics
310:                INCR n.total%(index%, harmonic%), frequency%(amplitude%, index%, harmonic%)
311:            NEXT
312:        NEXT
313:  NEXT
314:
315:  harmonic% = 0
316:  WHILE (harmonics%(harmonic%)<>0) AND (harmonic<=%num.harmonics)
317:
318:        IF (num.files%=1) THEN
319:            COLOR %option.color
320:            FOR index% = 0 TO 1
321:                SCREEN ,,index%,index%
322:                CLS
323:                FOR index.2% = %max.amplitude TO 0 STEP -1
324:                    PRINT TAB(%ati); USING "#.## |";index.2%*amp.step!
325:                    PRINT TAB(%ati);       "     |"
326:                NEXT
327:                LOCATE CSRLIN-2, 1+%ati
328:                PRINT TAB(%ati);           "     L_";
329:                FOR index.2% = 1 to 33
```

```
330:                    PRINT "⊥";
332:                NEXT
333:                PRINT
334:                PRINT TAB(%ati); "^Amplitude                    Trial # =>";
335:            NEXT
336:
337:            OPEN data.file.name$ + LEFT$(file.names$, INSTR(file.names$, " ")-1) + ".DAT" FOR INPUT AS #1
338:            FOR index% = 1 to %header.size      'skip header information
339:                INPUT #1, temp$
340:            NEXT
341:
342:                '*** skip thru file until correct harmonic is found
343:            INPUT #1, temp$
344:            index.2% = VAL(temp$)
345:            WHILE (NOT EOF(1)) AND (index.2%<>harmonics%(harmonic%))
346:                INPUT #1, temp$
347:                index.2% = VAL(temp$)
348:            WEND                                                'next line
349:
350:            COLOR %menu.color
351:            done% = (temp$="")
352:            response.1% = 0                                     'trial # of upper staircase
353:            response.2% = 0                                     'trial # of lower staircase
354:            WHILE (NOT done%) AND (NOT EOF(1))
355:                temp$ = MID$(temp$, 3)                          'skip harmonic number
356:                amplitude% = VAL(temp$)                         'save amplitude step number
357:                temp$ = MID$(temp$, 3)                          'skip amplitude info
358:                staircase% = VAL(temp$)                         'save staircase
359:                temp$ = MID$(temp$, 3)                          'skip staircase info
360:                index% = VAL(temp$)                             'get response code
361:                SCREEN ,,staircase%,staircase%                  'change to appropriate screen
362:                IF (staircase%=0) THEN                          'increment appropriate counter
363:                    INCR response.1%
364:                    LOCATE (%max.amplitude-amplitude%)*2 + 1, 5+%ati+response.1%
365:                ELSE
366:                    INCR response.2%
367:                    LOCATE (%max.amplitude-amplitude%)*2 + 1, 5+%ati+response.2%
368:                END IF
369:                IF (index%=1) THEN                              'plot response
370:                    PRINT "C";
371:                ELSE
372:                    PRINT "I";
373:                END IF
374:                INPUT #1, temp$                                 'get a line of data
375:                done% = (temp$="")                              'end of harmonic?
376:            WEND                                                'next line
377:            CLOSE #1                                            'close input file
378:        END IF
379:
380:                '*** determine which response to use
381:        IF ((n.total%(0, harmonic%) < n.total%(1, harmonic%)) AND _     'if # incorrect responses < # correct responses
382:           (n.total%(0, harmonic%) <> 0)) THEN
383:            response.1% = 0                                     'use incorrect responses
384:        ELSE

FILE=fourier4.bas    Wed Jun 14 16:23:01 1989    PAGE=7
```

```
385:            response.1% = 1                                                 'use correct responses
387:        END IF
388:
389:            '*** calculate (Yi * Ni) and (Yi^2 * Ni)
390:        FOR amplitude% = 0 TO %max.amplitude
391:            table!(amplitude%,0,harmonic%) = amplitude% * amp.step! * frequency%(amplitude%,response.1%,harmonic%)
392:            table!(amplitude%,1,harmonic%) = (amplitude%*amp.step!)^2 * frequency%(amplitude%,response.1%,harmonic%)
393:        NEXT
394:
395:            '*** calculate (Yi * Ni) and (Yi^2 * Ni) totals
396:        yn.total!  = 0          'reset totals
397:        y2n.total! = 0
398:        FOR amplitude% = 0 TO %max.amplitude
399:            INCR yn.total!, table!(amplitude%,0,harmonic%)
400:            INCR y2n.total!, table!(amplitude%,1,harmonic%)
401:        NEXT
402:
403:            '*** estimate mean and standard deviation and print results
404:        COLOR %menu.color
405:        SCREEN ,,0,0
406:        LOCATE (%max.amplitude*2+3), 1+%ati
407:        PRINT TAB(%ati); "FILENAME: ";file.names$, "UPPER STAIRCASE"
408:        PRINT TAB(%ati); "HARMONIC = ";harmonics%(harmonic%)
409:        IF (response.1% = 0) THEN
410:            est.mean! = yn.total! / n.total%(response.1%, harmonic%) + .5 * amp.step!
411:        ELSE
412:            est.mean! = yn.total! / n.total%(response.1%, harmonic%) - .5 * amp.step!
413:        END IF
414:        sy2! = (y2n.total! - yn.total!^2 / n.total%(response.1%, harmonic%)) / (n.total%(response.1%, harmonic%)-1)
415:        est.deviation! = 1.620*amp.step!*(sy2!/(amp.step!^2)+0.029)
416:        IF (n.total%(0, harmonic%) = 0) THEN
417:            PRINT TAB(%ati); USING "    estimated mean=###.####         estimated standard deviation=###.#####"; _
418:                             est.mean!; est.deviation!;
419:        ELSE
420:            PRINT TAB(%ati); USING "    estimated mean=###.####         estimated standard deviation=###.#####"; _
421:                             est.mean!; est.deviation!;
422:        END IF
423:
424:        COLOR %menu.color
425:        SCREEN ,,1,1
426:        IF (num.files%>1) THEN CLS
427:        LOCATE (%max.amplitude*2+3), 1+%ati
428:        IF (num.files%=1) THEN
429:            PRINT TAB(%ati); "FILENAME: ";file.names$, "LOWER STAIRCASE"
430:        END IF
431:        PRINT TAB(%ati); "HARMONIC = ";harmonics%(harmonic%)
432:        IF (n.total%(0, harmonic%) = 0) THEN
433:            PRINT TAB(%ati); USING "    estimated mean=###.####         estimated standard deviation=###.#####"; _
434:                             est.mean!; est.deviation!;
435:        ELSE
436:            PRINT TAB(%ati); USING "    estimated mean=###.####         estimated standard deviation=###.#####"; _
437:                             est.mean!; est.deviation!;
438:        END IF
439:        PRINT #2, USING "    ##    | ##.#### |  ##.##### | ##.#### | ##.#### | ##.#### "; _
```

```
                        harmonics%(harmonic%), 0.5-est.mean!, est.deviation!/SQR(n.total%(response.1%, harmonic%)),_
                        est.mean!, n.total%(response.1%, harmonic%), right%(harmonic%)/num.responses%(harmonic%)*100

        IF (num.files%=1) THEN
            SCREEN ,,0,0
            IF print.scrn% THEN
                CALL INTERRUPT 5                                    'print screen
                LPRINT : LPRINT : LPRINT                            'space
                DELAY 0.5                                           'wait for printer
            END IF
            IF NOT file.only% THEN
                COLOR %error.color
                LOCATE 25, 1+%ati
                PRINT "PRESS ANY KEY TO CONTINUE";
                CALL get.key(temp$)
            END IF
        END IF
        SCREEN ,,1,1
        IF print.scrn% THEN
            CALL INTERRUPT 5                                        'print screen
            LPRINT CHR$(12);                                        'form feed
            DELAY 4                                                 'wait for printer
        END IF
        IF NOT file.only% THEN
            COLOR %error.color
            LOCATE 25, 1+%ati
            PRINT "PRESS ANY KEY TO CONTINUE";
            CALL get.key(temp$)
        END IF

        INCR harmonic%                                              'next harmonic
    WEND

    CLOSE #2                                                        'close output file

END SUB




'┌──────────────────────────────────────────────────────────────────────┐
'│    BEGIN.TESTING                                                     │
'│                                                                      │
'│        This module is used to display the chosen stimuli and         │
'│    collect the experimental data.  See FOURIER4.DOC for more         │
'│    detailed information.                                             │
'└──────────────────────────────────────────────────────────────────────┘

SUB begin.testing

        ' **** VARIABLE DECLARATIONS
    LOCAL amplitude%            'current amplitude
```

```
495:     LOCAL block%                'number of harmonics displayed in session
497:     LOCAL file.name$            'filename of screen to be loaded
498:     LOCAL harmonic%             'harmonic number in use
499:     LOCAL last.case%            'last staircase used (0=upper, -1=lower)
500:     LOCAL low.case%             'amplitude to use next for lower staircase
501:     LOCAL low.response%         'last response for lower staircase
502:     LOCAL number.of.harmonic%   'number of lines in data file
503:     LOCAL num.case%             'number of times either case has been repeated
504:     LOCAL num.trials%           'number of reversals
505:     LOCAL response%             'subject's response
506:     LOCAL reversed%             'whether a reversal has occurred or not
507:     LOCAL side%                 'side of screen nonfundamental is put on
508:     LOCAL stair.case%           'current staircase
509:     LOCAL temp$                 'temporary string variable
510:     LOCAL up.case%              'amplitude to use next for upper staircase
511:     LOCAL up.response%          'last response for upper staircase
512:
513:
514:         ' **** GLOBAL DECLARATIONS
515:     SHARED demo.flag%
516:     SHARED escape$                      'character string returned by escape key
517:     SHARED image.file.name$             'file name of screen images (.SCR)
518:     SHARED image.size$                  'size of images in mm
519:     SHARED random.file.name$            'file name of random harmonic data
520:     SHARED sample.image$                'file name prefix of sample harmonic
521:     SHARED temp.file.name$              'file name of temporary data
522:
523:
524:     CLS
525:     COLOR %menu.color
526:     PRINT TAB(24);"FOURIER 4: DATA COLLECTION SCREEN"
527:
528:     OPEN random.file.name$ FOR INPUT AS #3
529:     INPUT #3, num.blocks%, number.of.harmonic%
530:     FOR block% = 2 to number.of.harmonic%               'read past used data
531:         INPUT #3, harmonic%, up.case%, low.case%
532:     NEXT
533:
534:     block% = 1
535:     WHILE (block% <= num.blocks%) AND (NOT EOF(3))
536:
537:         INPUT #3, harmonic%, up.case%, low.case%
538:         RANDOMIZE TIMER
539:         up.response%  = -1      'set last response, to upper case, to illegal value
540:         low.response% = -1      'set last response, to lower case, to illegal value
541:         last.case%    = -2      'set last used to illegal value
542:         num.case%     =  0      'reset number of repeats of case
543:         num.trials%   =  0      'reset number of reversals
544:         reversed%     = %FALSE  'no reversal has occurred yet
545:
546:                 '*** display blank screen, sample harmonic, blank screen
547:         CALL PGA.transmit("DI,0 L,1,2,2,5 L,2,2,2,5 L,3,2,2,5 L,4,2,2,5 ")
548:         CALL PGA.transmit("L,5,2,2,5 L,6,2,2,5 L,7,2,2,5 L,8,2,2,5 ")   'blank screen
549:         CALL load.noise.screen
```

```
550:        temp$ = STR$(harmonic%)
552:        IF (harmonic%<10) THEN
553:            temp$ = "0" + RIGHT$(temp$, LEN(temp$)-1)
554:        ELSE
555:            temp$ = RIGHT$(temp$, LEN(temp$)-1)
556:        END IF
557:        file.name$ = image.file.name$ + image.size$ + temp$ + sample.image$ + "L.IMG"
558:        CALL PGA.transmit("MK,2 ")
559:        CALL load.screen(file.name$)
560:        file.name$ = image.file.name$ + image.size$ + temp$ + "OR.IMG"
561:        CALL load.screen(file.name$)
562:        BEEP
563:        CALL get.response(response%)    'wait for subject to press a key
564:        CALL PGA.transmit("L,2,5,6,11 L,3,5,6,11 L,6,5,6,11 L,7,5,6,11 L,8,5,6,11 W,120 ")      'display sample harmonic
565:        IF demo.flag% THEN
566:            CALL get.response(response%)
567:        END IF
568:        CALL PGA.transmit("L,2,2,2,5 L,3,2,2,5 L,6,2,2,5 L,7,2,2,5 L,8,2,2,5 ")                 'display blank screen
569:        CALL get.response(response%)    'wait for response
570:        CALL PGA.transmit("L,1,6,6,12 L,3,6,6,12 L,5,6,6,12 L,7,6,6,12 L,8,6,6,12 ")            'display noise screen
571:
572:        WHILE (num.trials% < %max.num.trials)
573:            IF (num.case% = %max.num.case) THEN
574:                stair.case% = NOT last.case%
575:                num.case%   = 0
576:            ELSE
577:                stair.case% = INT(RND*2)-1              'get staircase
578:                IF (stair.case% = last.case%) THEN
579:                    INCR num.case%
580:                ELSE
581:                    num.case%  = 0
582:                END IF
583:            END IF
584:            IF (stair.case%=0) THEN                     'get amplitude number
585:                amplitude% = up.case%
586:            ELSE
587:                amplitude% = low.case%
588:            END IF
589:            side% = INT(RND*2)                          'get side
590:            temp$ = STR$(harmonic%)
591:            IF (harmonic%<10) THEN
592:                temp$ = "0" + RIGHT$(temp$, LEN(temp$)-1)
593:            ELSE
594:                temp$ = RIGHT$(temp$, LEN(temp$)-1)
595:            END IF
596:            file.name$ = image.file.name$ + image.size$ + temp$
597:            CALL PGA.transmit("MK,2 ")
598:            IF (side%=0) THEN
599:               file.name$ = file.name$ + CHR$(amplitude% + ASC("0")) + "L.IMG"
600:               CALL load.screen(file.name$)     'load image on left side
601:               file.name$ = LEFT$(file.name$,2+LEN(image.file.name$)+LEN(image.size$)) + "OR.IMG"
602:               CALL load.screen(file.name$)     'load fundamental on right side
603:            ELSE
604:               file.name$ = file.name$ + CHR$(amplitude% + ASC("0")) + "R.IMG"
```

```
605:                CALL load.screen(file.name$)     'load image on right side
607:                file.name$ = LEFT$(file.name$,2+LEN(image.file.name$)+LEN(image.size$)) + "OL.IMG"
608:                CALL load.screen(file.name$)     'load fundamental on left side
609:             END IF
610:             IF demo.flag% THEN
611:                 CALL demo.screens
612:             ELSE
613:                 CALL flash.screens
614:             END IF
615:             CALL get.response(response%)                'get subject response
616:             PRINT #1, USING "## ";harmonic%;            'write harmonic number
617:             PRINT #1, USING "# ";amplitude%;            'write amplitude number
618:             IF (stair.case% = 0) THEN                   'write staircase
619:                 PRINT #1, "0 ";                         'upper
620:             ELSE
621:                 PRINT #1, "1 ";                         'lower
622:             END IF
623:             IF  (amplitude% <> 0) AND _
624:                 (((response% = %left.button) AND (side% = 0)) OR _
625:                 ((response% = %right.button) AND (side% = 1))) THEN
626:                 PRINT #1, "1 ";                         'correct
627:             ELSE
628:                 PRINT #1, "0 ";                         'incorrect
629:             END IF
630:             IF (response% = %left.button) THEN          'write response
631:                 PRINT #1, "0";                          'left
632:             ELSE
633:                 PRINT #1, "1";                          'right
634:             END IF
635:             PRINT #1, side%
636:             IF (stair.case% = 0) THEN
637:                 IF ((response% = %left.button) AND (side% = 1)) OR _
638:                    ((response% = %right.button) AND (side% = 0)) OR _
639:                    (amplitude% = 0) THEN
640:                     IF (up.case%<%max.amplitude) THEN INCR up.case%
641:                     reversed% = (up.response%=0)        'reverse=change in direction
642:                     up.response% = 1                    'going up now
643:                 ELSE
644:                     IF (up.case%>%min.amplitude) THEN DECR up.case%
645:                     reversed% = (up.response%=1)        'reverse=change in direction
646:                     up.response% = 0                    'going down now
647:                 END IF
648:             ELSE
649:                 IF ((response% = %left.button) AND (side% = 1)) OR _
650:                    ((response% = %right.button) AND (side% = 0)) OR _
651:                    (amplitude% = 0) THEN
652:                     IF (low.case%<%max.amplitude) THEN INCR low.case%
653:                     reversed% = (low.response%=0)
654:                     low.response% = 1
655:                 ELSE
656:                     IF (low.case%>%min.amplitude) THEN DECR low.case%
657:                     reversed% = (low.response%=1)
658:                     low.response% = 0
659:                 END IF
```

```
660:            END IF
662:         last.case% = stair.case%                  'save current staircase
663:         IF reversed% THEN INCR num.trials%        'increment number of reversals
664:         IF (((response% = %left.button) AND (side% = 1)) OR
665:            ((response% = %right.button) AND (side% = 0))) AND _
666:            (amplitude% <> 0) THEN
667:             PLAY "L16N1"
668:         END IF
669:         IF INKEY$=escape$ THEN            'stop if ESCape key pressed
670:            CALL PGA.transmit("CA DI,1 ")
671:            PRINT #1,
672:            PRINT #1, "COMMENTS: "
673:            PRINT #1, "            Terminated early by user."
674:            CLOSE #1
675:            CLOSE #3
676:            EXIT SUB
677:         END IF
678:         WEND     'trial
679:         PRINT #1,
680:         block% = block% + 1
681:      WEND        'block
682:
683:      CALL pga.transmit("CA DI,1 ")
684:      PRINT
685:      PRINT
686:      PRINT TAB(%entry.indent);
687:      COLOR %option.color
688:      INPUT "ENTER COMMENTS: ";temp$
689:      PRINT #1,
690:      PRINT #1, "COMMENTS: "
691:      PRINT #1, "            "; temp$
692:      CLOSE #1   'close data file
693:      CLOSE #3   'close input file
694:
695: END SUB
696:
697:
698:
699:
700: '
701: '    DEMO.SCREENS
702: '
703: '
704: '        This module displays the adapting screen if necessary, then
705: '     displays the noise screen, then the stimulus screen, and returns
706: '     to the noise screen and the calling module.  A beep is sounded in
707: '     between the adapting screen and noise screen.  (If the adapting
708: '     screen is not to be shown, the beep is sounded immediately.)  The
709: '     adapting screen is shown for 1000 ms.  The noise is displayed for
710: '     1000 ms if the adapting screen has been displayed or for 1000 ms
711: '     if the adapting screen has not been displayed.  The stimulus is
712: '     displayed for 150 ms.  NOTE: in between each screen display, the
713: '     module waits for the user to press a response key.
714: '
```

```
'└──────────────────────────────────────────────────────────────────────┘

SUB demo.screens

        ' **** LOCAL DECLARATIONS
    LOCAL  response%

        ' **** GLOBAL DECLARATIONS
    SHARED adapt.flag%
    SHARED demo.flag%


    IF adapt.flag% THEN
        CALL get.response(response%)
        'show adapting harmonic
        CALL PGA.transmit("L,1,2,2,5 L,3,2,2,5 L,4,5,6,11 L,5,5,6,11 L,6,5,6,11 L,7,5,6,11 W,120 ")
        CALL get.response(response%)
        'show noise screen
        CALL PGA.transmit("L,1,6,6,12 L,3,6,6,12 L,4,2,2,5 L,5,6,6,12 L,6,2,2,5 L,7,6,6,12 W,60 ")
        CALL get.response(response%)
    ELSE
        'show noise screen
        CALL PGA.transmit("W,60 ")
        CALL get.response(response%)
    END IF

    'show stimulus
    CALL PGA.transmit("L,1,2,2,5 L,2,5,6,11 L,3,5,6,11 L,5,2,2,5 L,6,5,6,11 L,7,5,6,11 L,8,5,6,11 W,9 ")
    CALL get.response(response%)

    'show noise
    CALL PGA.transmit("L,1,6,6,12 L,2,2,2,5 L,3,6,6,12 L,5,6,6,12 L,6,2,2,5 L,7,6,6,12 ")

END SUB  'demo.screens



'┌──────────────┐
'│ DISPLAY.MENU │
'└──────────────┴──────────────────────────────────────────────────────┐
'                                                                      │
'       This module displays the menu options, gets a keystoke, and    │
'    executes the appropriate module.  The keystroke is returned to    │
'    the main program.                                                 │
'                                                                      │
'└──────────────────────────────────────────────────────────────────────┘

SUB display.menu(response$)

    SHARED      demo.flag%
    SHARED      escape$

                '*** setup the screen
```

```
770:     CLS
772:     COLOR %menu.color
773:     PRINT TAB(31); "FOURIER 4: MAIN MENU"
774:     PRINT
775:     PRINT
776:     PRINT
777:     CALL print.option("I|INITIALIZE PGA")
778:     CALL print.option("R|RANDOMIZE TRIAL DATA")
779:     CALL print.option("E|ENTER SUBJECT DATA")
780:     CALL print.option("D|DEMO")
781:     CALL print.option("C|COLLECT DATA")
782:     CALL print.option("A|ANALYZE DATA")
783:     CALL print.option("P|PLOT SUM FILE")
784:     CALL print.option("S|ENTER STIMULUS INFO");
785:     CALL print.option("T|TEST RESPONSE BOX")
786:     CALL print.option("X|EXIT PROGRAM")
787:     CALL print.option("<ESC>|<ESC> TERMINATE PROGRAM")
788:     PRINT
789:     PRINT
790:     PRINT TAB(%entry.indent); "ENTER OPTION: ";
791:
792:                 '*** get the user's option
793:     CALL get.key(response$)
794:     PRINT response$;                                'echo keypress
795:
796:                 '*** execute option
797:     SELECT CASE UCASE$(response$)
798:         CASE "A"                                    'analyze data
799:             CALL analyze.data
800:         CASE "C"                                    'data collection
801:             IF NOT data.flag% THEN
802:                 CALL print.error("ERROR: subject data not loaded yet.")
803:             ELSE
804:                 demo.flag% = %false
805:                 CALL begin.testing
806:                 data.flag% = %false
807:             END IF
808:         CASE "D"
809:             IF NOT data.flag% THEN
810:                 CALL print.error("ERROR: subject data not loaded yet.")
811:             ELSE
812:                 demo.flag% = %true
813:                 CALL begin.testing
814:                 data.flag% = %false
815:             END IF
816:         CASE "E"                                    'enter subject data
817:             CALL enter.subject.data
818:             data.flag% = %true
819:         CASE "I"                                    'initialize the PGA
820:             CALL initialize.pga
821:             noise.flag% = %false
822:         CASE "P"                                    'plot a .OTS file
823:              CALL plot.sum.file
824:         CASE "R"                                    'randomize the trial data
```

```
825:            CALL randomize.trial.data
827:        CASE "S"                                'make harmonic info
828:            CALL confirm(response$)
829:            IF UCASE$(response$) = "Y" THEN
830:                CALL make.random.info
831:            END IF
832:        CASE "T"                                'test response box
833:            CALL test.response.box
834:        CASE "X", escape$                       'exit the program
835:            CALL exit.program
836:            response$ = "X"
837:        CASE ELSE                               'invalid keystroke
838:            BEEP
839:    END SELECT
840:
841: END SUB
842:
843:
844:
845:
846: '
847: '  ENTER.SUBJECT.DATA
848: '
849: '
850: '       This module reads the subject's data, opens the output file
851: '      ("XXX\YYYYZZZZ.DAT" where XXX is the data directory, YYYY is
852: '      the subject's initials and ZZZZ is the session number) and
853: '      saves the data to it (with the current date).  The file is
854: '      left open.  The user is asked to confirm that he has correctly
855: '      entered the data.  If he presses any key other than Y, the file
856: '      will be closed and erased and the user will be asked for the
857: '      subject's data again.  This module also loads in the adapting
858: '      stimulus screen if the number entered for the acapting harmonic
859: '      is not zero.
860: '
861: '
862:
863: SUB enter.subject.data
864:
865:        ' **** VARIABLE DECLARATIONS
866:    LOCAL good%          'boolean used to determine if data was entered correctly
867:    LOCAL file.name$     'filename of data output file
868:    LOCAL string.1$      'general string variable
869:    LOCAL string.2$      'general string variable
870:
871:        ' *** GLOBAL VARIABLE DECLARATIONS
872:    SHARED adapt.flag%           'adapting stimulus operation flag
873:    SHARED adapt.file.name$      'prefix of adapting stimulus files
874:    SHARED data.file.name$       'prefix of data files
875:    SHARED fix.file.name$        'fixation point file name
876:    SHARED image.file.name$      'prefix of image files
877:    SHARED image.size$           'size of image in mm
878:
879:    DO
```

```
880:
882:        good% = %false          'assume entries are incorrect
883:
884:                '*** setup screen
885:        CLS
886:        COLOR %menu.color
887:        PRINT TAB(23); "FOURIER 4: SUBJECT DATA ENTRY SCREEN"
888:        PRINT
889:        PRINT
890:        COLOR %option.color
891:
892:                '*** get data to create filename
893:        PRINT TAB(%entry.indent); : INPUT "SUBJECT INITIALS: ", string.1$
894:        PRINT TAB(%entry.indent); : INPUT "SESSION NUMBER: ", string.2$
895:
896:                '*** determine filename and open data file
897:        file.name$ = data.file.name$ + string.1$ + string.2$ + ".DAT"
898:        OPEN file.name$ FOR OUTPUT AS #1
899:
900:                '*** save header in data file and print data
901:        PRINT #1, "FOURIER 4 Data file: "; file.name$,, "Created: "; date$
902:        PRINT #1,
903:        PRINT #1, "SUBJECT INITIALS: "; string.1$
904:        PRINT #1, "SESSION NUMBER: "; string.2$
905:
906:                '*** get rest of data and save it
907:        PRINT TAB(%entry.indent); : INPUT "STANDARD: ", string.1$
908:        PRINT #1, "STANDARD: "; string.1$
909:
910:                '*** get adapting harmonic and load screens if necessary
911:        PRINT TAB(%entry.indent); : INPUT "ADAPTING HARMONIC: ", string.1$
912:        PRINT #1, "ADAPTING HARMONIC: "; string.1$
913:        IF string.1$<>"0" THEN                                          'load screen
914:            CALL PGA.transmit("MK,4 ")                                  'set mask
915:            CALL load.screen(adapt.file.name$+string.1$+"L.IMG")        'load in the adapting harmonic screen
916:            CALL load.screen(adapt.file.name$+string.1$+"R.IMG")        'load in the adapting harmonic screen
917:            CALL PGA.transmit("MK,8 ")
918:            CALL load.screen(fix.file.name$)
919:            adapt.flag% = %true
920:        END IF
921:        COLOR %option.color
922:        PRINT TAB(%entry.indent);
923:        INPUT "ENTER TEST STIMULUS SIZE (1st 3 chars of image set): ", image.size$
924:        PRINT #1, "TEST STIMULUS SIZE: "; image.size$
925:        PRINT #1,
926:
927:        PRINT #1, "  HARMONIC # | AMPLITUDE # |    STAIRCASE    |      CORRECT      |    RESPONSE    |     SIDE"
928:        PRINT #1, "2 4 6 8 12 16 |     0-8     | 0=upper 1=lower | 0=wrong 1=correct | 0=left 1=right | 0=left 1=right"
929:        PRINT #1,
930:        PRINT #1,
931:
932:                '*** make sure everything is OK
933:        CALL confirm(string.1$)
934:        IF UCASE$(string.1$) = "Y" THEN         'everything OK?
```

```
            good% = %true                          ' yes.
         ELSE                                      'no, so:
            CLOSE #1                               'close the file
            KILL file.name$                        'and erase it
         END IF

      LOOP UNTIL good%              'repeat until everything is OK

END SUB



'
'  ┌──────────────┐
'  │ EXIT.PROGRAM │
'  └──────────────┘
'
'       This module is called at program termination.  It resets
'     the screen back to CGA emulation mode, closes all files, and
'     prints a termination message.
'
'

SUB exit.program

    CLS
    CALL PGA.transmit("CA DI,1 ")
    CLOSE
    PRINT "PROGRAM TERMINATED."

END SUB



'
'  ┌──────────────┐
'  │ FLASH.SCREENS │
'  └──────────────┘
'
'       This module displays the adapting screen if necessary, then
'     displays the noise screen, then the stimulus screen, and returns
'     to the noise screen and the calling module.  A beep is sounded in
'     between the adapting screen and noise screen.  (If the adapting
'     screen is not to be shown, the beep is sounded immediately.)  The
'     duration of the adapting, noise, and stimulus screens is
'     determined by the number (#) of screen refreshes specified by
'     "W,#".
'

SUB flash.screens

        ' **** GLOBAL DECLARATIONS
    SHARED adapt.flag%
```

```
 990:
 992:
 993:     IF adapt.flag% THEN
 994:         'show adapting harmonic
 995:         CALL PGA.transmit("L,1,2,2,5 L,3,2,2,5 L,4,5,6,11 L,5,5,6,11 L,6,5,6,11 W,120 ")
 996:         'show noise screen
 997:         CALL PGA.transmit("L,1,6,6,12 L,3,6,6,12 L,4,2,2,5 L,6,2,2,5 W,60 ")
 998:     ELSE
 999:         'show noise screen
1000:         CALL PGA.transmit("W,60 ")
1001:     END IF
1002:
1003:     'show stimulus
1004:     CALL PGA.transmit("L,1,2,2,5 L,2,5,6,11 L,3,5,6,11 L,5,2,2,5 L,6,5,6,11 L,7,5,6,11 W,9 ")
1005:
1006:     'show noise
1007:     CALL PGA.transmit("L,1,6,6,12 L,2,2,2,5 L,3,6,6,12 L,5,6,6,12 L,6,2,2,5 L,7,6,6,12 ")
1008:
1009: END SUB  'flash.screens
1010:
1011:
1012:
1013:
1014: '
1015: '   INITIALIZE.PGA
1016: '
1017: '
1018: '          This module initializes the PGA to a known state, i.e., it
1019: '      sets the communication mode to ASCII, clears the buffers, resets
1020: '      the flags, defines the LUT entries, sets the mask, clears the
1021: '      screen, and sets the current color.
1022: '
1023: '
1024:
1025: SUB initialize.pga
1026:
1027:         ' **** VARIABLE DECLARATIONS
1028:     LOCAL in.string$    'temporary string
1029:
1030:
1031:         ' *** restart the PGA, enable errors, and clear the buffers
1032:     CALL PGA.cold.restart
1033:     DELAY 0.1
1034:     CALL PGA.error.enable
1035:     CALL PGA.recieve(in.string$)                'clear buffer
1036:     CALL PGA.error.recieve(in.string$)          'clear buffer
1037:
1038:         ' *** set communications to ASCII and reset the flags
1039:     CALL PGA.transmit("CA RF WI,-320,320,-240,240 ")
1040:
1041:         ' *** define LUT values (noise is set to on)
1042:     CALL PGA.transmit("LI,5 L,0,2,2,5 L,1,6,6,12 L,2,2,2,5 L,3,6,6,12 ")
1043:     CALL PGA.transmit("L,4,2,2,5 L,5,6,6,12 L,6,2,2,5 L,7,6,6,12 L,8,6,6,12 ")
1044:
```

```
1045:         ' *** clear the screen
1047:     CALL PGA.transmit("CLS,0 ")
1048:     CALL PGA.error.recieve(in.string$)
1049:     IF in.string$ <> "" THEN CALL print.error("PGA ERROR: "+in.string$)
1050:
1051: END SUB
1052:
1053:
1054:
1055:
1056: '
1057: '   LOAD.NOISE.SCREEN
1058: '
1059: '
1060: '         This module randomly selects one of the noise (masking)
1061: '       screens for use with one block of trials (the number of blocks
1062: '       having been chosen using the "Enter Stimulus Info" menu option).
1063: '       The number of noise screens to choose from is determined by the
1064: '       variable "num.noise" and must correspond to the number of
1065: '       noise-screen files on disk and named "XXXX#.SCR" where XXXX is
1066: '       the string "noise.file.name$" and # is a number between 1 and
1067: '       "num.noise".  The image is loaded into the PGA's memory.
1068: '       This process begins when "Collect Data" is chosen from the menu
1069: '       and takes about 6 seconds.  This module also reads in the
1070: '       fixation point from the file fix.file.name$.
1071: '
1072: '
1073:
1074: SUB load.noise.screen
1075:         ' **** GLOBAL DECLARATIONS
1076:     SHARED fix.file.name$       'file name of fixation point
1077:     SHARED noise.file.name$     'file name prefix of noise screen
1078:
1079:         ' **** LOCAL VARIABLE DECLARATIONS
1080:     LOCAL  num%                 'number of noise screen to load in
1081:
1082:         ' *** determine the filename, set the mask, and load the noise screen
1083:     num% = INT(RND*%num.noise) + 1
1084:     CALL PGA.transmit("MK,1 ")
1085:     CALL load.screen(noise.file.name$ + CHR$(ASC("0")+num%) + ".SCR")
1086:
1087:         ' *** load the fixation point
1088:     CALL PGA.transmit("MK,8 ")
1089:     CALL load.screen(fix.file.name$)
1090:
1091: END SUB
1092:
1093:
1094:
1095:
1096: '
1097: '   LOAD.SCREEN
1098: '
1099: '
```

```
1100: '┌──────────────────────────────────────────────────────────────────────┐
1102: '│       This module loads in a screen (in hex format) from the file    │
1103: '│     specified by its only parameter.                                 │
1104: '└──────────────────────────────────────────────────────────────────────┘
1105:
1106: SUB load.screen(file.name$)
1107:
1108:         ' **** VARIABLE DECLARATIONS
1109:     LOCAL file.size&     'size of screen file in bytes
1110:     LOCAL in.string$     'temporary string
1111:
1112:         ' *** open file and get size
1113:     OPEN file.name$ FOR BINARY AS #2
1114:     file.size& = LOF(2)                                          'get file size
1115:     IF file.size& = 0 THEN
1116:        CALL pga.transmit("CA DI,1 ")
1117:        CALL print.error("ERROR: "+file.name$+" not found.")
1118:        CLOSE #2
1119:        ERROR 255
1120:     END IF
1121:
1122:         ' *** load in screen and transmit to PGA
1123:     SEEK #2, 0                                                   'goto beginning of file
1124:
1125:     CALL PGA.error.recieve(in.string$)                           'clear error buffer
1126:     IF in.string$ <> "" THEN CALL print.error("PGA ERROR: "+in.string$)
1127:     CALL PGA.transmit("CX ")                                     'switch to hex mode
1128:
1129:     COLOR %menu.color
1130:     PRINT TAB(%entry.indent); "READING: ";
1131:     WHILE NOT EOF(2)
1132:         IF file.size& < %buffer.size THEN                        'if less bytes to read than buffer size
1133:             GET$ #2, file.size&, in.string$                      'read in the number left
1134:         ELSE                                                     'otherwise
1135:             GET$ #2, %buffer.size, in.string$                    'read in a bufferfull
1136:             file.size& = file.size& - %buffer.size               'and adjust number of bytes left to be read
1137:         END IF
1138:         PRINT ". ";
1139:         CALL PGA.transmit(in.string$)                            'send to the PGA
1140:         CALL PGA.error.recieve(in.string$)                       'check for errors
1141:         IF in.string$ <> "" THEN
1142:             CALL print.error("PGA ERROR NUMBER: "+STR$(ASC(in.string$)))
1143:         END IF
1144:     WEND
1145:
1146:     CALL PGA.transmit("CA ")                                     'return to ASCII mode
1147:
1148:     CLOSE #2                                                     'close the data file
1149:
1150: END SUB
1151:
1152:
1153:
1154:
```

```
1155: '
1157: '   MAKE.RANDOM.INFO
1158: '
1159: '
1160: '          This module creates the information used by the randomize
1161: '       (harmonics) procedure.  Specifically, the user is asked for
1162: '       the harmonics and their respective starting positions for each
1163: '       of the staircases.
1164: '
1165: '
1166:
1167: SUB make.random.info
1168:
1169:         '**** GLOBAL VARIABLE DECLARATIONS
1170:     SHARED      random.file.name$       'file name of random harmonic info
1171:
1172:         '**** LOCAL VARIABLE DECLARATIONS
1173:     LOCAL       harmonic%               'number of harmonic
1174:     LOCAL       low.start%              'starting place of lower staircase
1175:     LOCAL       num%                    'number of harmonics in file
1176:     LOCAL       up.start%               'starting place of upper staircase
1177:
1178:     CLS
1179:     COLOR %menu.color
1180:     PRINT TAB(19); "FOURIER 4: MAKE RANDOM HARMONIC INFO SCREEN"
1181:     PRINT
1182:     PRINT
1183:     COLOR %option.color
1184:
1185:     OPEN random.file.name$ FOR OUTPUT AS #2
1186:     PRINT TAB(%entry.indent); : INPUT "NUMBER OF HARMONICS TO BE TESTED: ";num%
1187:     WRITE #2, num%, 1
1188:     FOR num% = 1 TO num%
1189:         PRINT TAB(%entry.indent);
1190:         INPUT "ENTER HARMONIC #, UPPER STAIRCASE START, LOWER STAIRCASE START: ";harmonic%, up.start%, low.start%
1191:         WRITE #2, harmonic%, up.start%, low.start%
1192:     NEXT
1193:
1194:     CLOSE #2
1195:
1196: END SUB
1197:
1198:
1199:
1200: '
1201: '   PLOT.SUM.FILE
1202: '
1203: '
1204: '          This module plots the means of each harmonic versus the
1205: '       harmonic number.  Data is read from the given sum file.
1206: '
1207: '
1208:
1209: SUB plot.sum.file
```

```
1210:
1212:           '*** LOCAL VARIABLE DECLARATIONS
1213:     LOCAL  filename$
1214:     LOCAL  harmonics%()                                    'harmonic numbers used
1215:     LOCAL  i
1216:     LOCAL  j
1217:     LOCAL  means!()                                        'means for harmonics
1218:     LOCAL  num.harmonics%
1219:     LOCAL  temp$
1220:
1221:
1222:           '*** GLOBAL VARIABLE DECLARATIONS
1223:       SHARED data.file.name$
1224:
1225:
1226:     DIM harmonics%(%num.harmonics)
1227:     DIM means!(%num.harmonics)
1228:
1229:
1230:           ' *** setup screen
1231:       CLS
1232:       COLOR %menu.color
1233:       PRINT TAB(28);"FOURIER 4: PLOT SUM FILE"
1234:       PRINT
1235:       PRINT
1236:
1237:           '*** get filename and open file
1238:       PRINT TAB(%entry.indent);
1239:       INPUT "FILENAME TO PLOT (.OTS) "; filename$
1240:       OPEN data.file.name$+filename$+".OTS" FOR INPUT AS #2
1241:
1242:           '*** skip header info
1243:       DO
1244:           INPUT #2, temp$
1245:       LOOP UNTIL (VAL(temp$)<>0)
1246:
1247:           '*** read data into arrays
1248:       num.harmonics% = 0
1249:       WHILE (temp$<>"")
1250:           INCR num.harmonics%
1251:           IF num.harmonics% > %num.harmonics THEN
1252:               CALL print.error("TOO MANY HARMONICS")
1253:               EXIT SUB
1254:           END IF
1255:           harmonics%(num.harmonics%) = VAL(temp$)
1256:           j = INSTR(temp$, "|")
1257:           j = INSTR(j+1, temp$, "|")
1258:           j = INSTR(j+1, temp$, "|")
1259:           means!(num.harmonics%) = VAL(MID$(temp$, j+1))
1260:           IF NOT EOF(2) THEN
1261:               INPUT #2, temp$
1262:           ELSE
1263:               temp$=""
1264:           END IF
```

```
    WEND

        '*** close file
    CLOSE #2

        '*** sort data
    FOR i = 1 TO num.harmonics%
       FOR j = 1 to i
           IF harmonics%(j) > harmonics%(i) THEN
                SWAP harmonics%(i), harmonics%(j)
                SWAP means!(i), means!(j)
           END IF
       NEXT
    NEXT

        '*** setup screen
    CLS
    COLOR %menu.color
    PRINT TAB(28);"FOURIER 4: PLOT SUM FILE"
    LOCATE 12, 1
    PRINT "MEAN";
    FOR i = 0 TO 10
        LOCATE 2+2*i, 6
        PRINT USING "#.## |"; i*.1
        PRINT TAB(11) "|"
    NEXT
    LOCATE 23, 11
    FOR i = 11 to 75
        PRINT "-";
    NEXT
    FOR i = 1 TO num.harmonics%
        LOCATE 24, 10+i*10
        PRINT USING "##"; harmonics%(i);
    NEXT
    LOCATE 25, 40
    PRINT "HARMONIC #";

        '*** plot data
    FOR i = 1 TO num.harmonics%
        LOCATE 2+2*means!(i)*10, 12+i*10
        COLOR %option.color
        PRINT "*";
        COLOR %menu.color
        PRINT USING " #.###"; means!(i);
    NEXT

        '*** wait for user to press key
    COLOR %error.color
    LOCATE 25, 1
    PRINT "PRESS ANY KEY";
    WHILE INKEY$=""
    WEND

END SUB
```

```
1320:
1322:
1323:
1324: '
1325: '  RANDOMIZE.TRIAL.DATA
1326: '
1327: '
1328: '          This module reads in the random number data file,
1329: '       randomizes the list, and writes the file back to disk.  The
1330: '       file is constructed so that the first number indicates the
1331: '       number of blocks in the file.  The next number represents the
1332: '       block to be used.  The remaining numbers indicate the harmonic
1333: '       number and its associated starting places for its upper stair-
1334: '       case and for its lower staircase.  The harmonics are read in
1335: '       sequential order and, thus, by randomizing that order, the
1336: '       order in which the harmonics are displayed is also randomized.
1337: '
1338: '
1339:
1340: SUB randomize.trial.data
1341:
1342:         ' **** VARIABLE DECLARATIONS
1343:     LOCAL file.pos%      'line number of next harmonic to be displayed
1344:     LOCAL file.size%     'size of file
1345:     LOCAL index%         'general indice
1346:     LOCAL index.2%       'used to swap values
1347:     LOCAL harmonics%()   'harmonic numbers
1348:     LOCAL up.case%()     'starts of upper staircases
1349:     LOCAL low.case%()    'starts of lower staircases
1350:
1351:
1352:         ' **** GLOBAL VARIABLE DECLARATIONS
1353:     SHARED random.file.name$    'filename of random number file
1354:
1355:
1356:         ' *** setup screen
1357:     CLS
1358:     COLOR %menu.color
1359:     PRINT TAB(25);"FOURIER 4: RANDOMIZE TRIAL DATA"
1360:     PRINT
1361:     PRINT
1362:
1363:     '*** open the data file, read in the number of elements, read in the
1364:     '*** elements, randomize them, and write them back to disk
1365:     PRINT TAB(%entry.indent); random.file.name$; ":", "READING...  ";
1366:     OPEN random.file.name$ FOR INPUT AS #2
1367:     INPUT #2, file.size%, file.pos%
1368:
1369:     DIM DYNAMIC harmonics%(file.size%), up.case%(file.size%), low.case%(file.size%)
1370:     FOR index% = 1 to file.size%
1371:         INPUT #2, harmonics%(index%), up.case%(index%), low.case%(index%)
1372:     NEXT
1373:
1374:     CLOSE #2
```

```
1375:
1377:     PRINT "RANDOMIZING...  ";
1378:     RANDOMIZE TIMER
1379:     FOR index% = 1 to file.size%                          ' for each element
1380:         index.2% = INT(RND*file.size%)+1                  ' pick a random element
1381:         SWAP harmonics%(index%), harmonics%(index.2%)     '  and swap
1382:         SWAP up.case%(index%), up.case%(index.2%)
1383:         SWAP low.case%(index%), low.case%(index.2%)
1384:     NEXT
1385:
1386:     PRINT "WRITING... "
1387:     OPEN random.file.name$ FOR OUTPUT AS #2
1388:     file.pos% = 1
1389:     WRITE #2, file.size%, file.pos%
1390:
1391:     FOR index% = 1 to file.size%
1392:         WRITE #2, harmonics%(index%), up.case%(index%), low.case%(index%)
1393:     NEXT
1394:
1395:     CLOSE #2
1396:
1397: END SUB
```

IV. LISTING OF AUXILIARY PROGRAMS

PGA.inc
PGAgraph.bas
PGAtran.asm
PGArecv.asm
PGAerrcv.asm
mkgrdata.bas
response.inc
toolbox.inc

```
1: '
2: '
3: '     FILENAME:       PGA.inc
4: '     PROGRAMMER:     Christopher Voltz - UDRI
5: '     CREATED:        -1/8706.18
6: '     LAST MOD!FIED:  -1/8801.11
7: '     TARGET:         IBM PC w/PGA
8: '     LANGUAGE:       Turbo BASIC v1.00
9: '     REQUIRED FILES: pgatran.bin, pgarecv.bin, pgaerrcv.bin,
10: '                    toolbox.inc
11: '
12: '           This file is the include module to provide
13: '    routines to use the professional graphics adapter
14: '    (PGA).  At the beginning of the program which wishes to
15: '    use these routines, enter the following lines:
16: '    $INCLUDE "toolbox.inc"
17: '    $INCLUDE "PGA.inc"
18: '
19: '           The following services are provided:
20: '    1) PGA.transmit ==> this routine transmits a string to
21: '       the PGA.  It is called with a string as a parameter.
22: '       eg.: CALL PGA.transmit(in.string$)  or
23: '            CALL PGA.transmit("CA ")
24: '    2) PGA.recieve ==> this routine reads data from the PGA
25: '       and places it in the string parameter sent to it.
26: '       A null string is returned if no data was available.
27: '       eg.: CALL PGA.recieve(out.string$)
28: '    3) PGA.error.recieve ==> this routine reads error
29: '       messages from the PGA.  A null string is returned
30: '       if no error message was available.
31: '       eg.: CALL PGA.error.recieve(out.string$)
32: '    4) PGA.error.enable ==> this routine sets the flag
33: '       which allows the PGA to send error messages.
34: '       eg.: CALL PGA.error.enable
35: '    5) PGA.error.disable ==> this routine resets the flag
36: '       which allows the PGA to send error messages.  When
37: '       reset, the PGA does not send error messages.
38: '       eg.: CALL PGA.error.disable
39: '    6) PGA.cold.restart ==> this routine causes the PGA to
40: '       perform a cold restart (ie. a hardware boot).
41: '       eg.: CALL PGA.cold.restart
42: '    7) PGA.warm.restart ==> this routine causes the PGA to
43: '       perform a warm restart (ie. a software boot).
44: '       eg.: CALL PGA.warm.restart
45: '    8) PGA.print.error ==> this routine switches to the
46: '       emulator screen, displays the error message and waits
47: '       for the user to press a key to indicate the message
48: '       was seen.
49: '       eg.: CALL PGA.print.error(out.string$) or
50: '            CALL PGA.print.error("error message")
51: '
52: '           The following identifiers are reserved:
53: '    %PGA.base, %PGA.error.flag, %PGA.cold.restart.flag,
54: '    %PGA.warm.restart.flag, %PGA.set, %PGA.reset,
```

FILE=pga.inc Wed Jun 14 16:19:25 1989 PAGE=1

```
55: '|      %PGA.max.string.                                              |
57: '|                                                                    |
58: '|            Note that it is the user's responsibility to            |
59: '|      send the correct initialization string ("CA" or "CX").        |
60: '|      Also note that if data is to be sent in the hexadecimal       |
61: '|      format, then the values must be placed in the string          |
62: '|      using the CHR$() function.  For further information           |
63: '|      regarding use of the PGA, refer to the IBM Technical          |
64: '|      Reference: Options and Adapters volume 3.                     |
65: '|                                                                    |
66: '|____________________________________________________________________|
67:
68:
69:
70: '|--------------------------------------------------------------------|
71: '|                      CONSTANT DEFINITIONS                          |
72: '|--------------------------------------------------------------------|
73:
74: %PGA.base       = &HC600                  'segment of PGA interface memory
75: %PGA.error.flag = &H0308                  'offset of error flag
76: %PGA.max.string =   1024                  'maximum string size returned
77: %PGA.cold.restart.flag = &H0306           'offset of cold restart flag
78: %PGA.warm.restart.flag = &H0307           'offset of warm restart flag
79: %PGA.set    = &HFF                       'code to set flag
80: %PGA.reset = &H00                        'code to reset flag
81:
82:
83:
84: '|--------------------------------------------------------------------|
85: '|                         SUBROUTINES                                |
86: '|--------------------------------------------------------------------|
87:
88: SUB PGA.cold.restart
89:     DEF SEG = %PGA.base                          'set segment
90:     POKE %PGA.cold.restart.flag, PGA.set         'set cold restart flag
91:     DEF SEG                                      'return to BASIC segment
92: END SUB
93:
94:
95: SUB PGA.error.enable
96:     DEF SEG = %PGA.base                          'set segment
97:     POKE %PGA.error.flag, %PGA.set               'set error enable flag
98:     DEF SEG                                      'return to BASIC segment
99: END SUB
100:
101:
102: SUB PGA.error.disable
103:     DEF SEG = %PGA.base                         'set segment
104:     POKE %PGA.error.flag, %PGA.reset            'reset error enable flag
105:     DEF SEG                                     'return to BASIC segment
106: END SUB
107:
108:
109: SUB PGA.error.recieve( out.string$ )
```

```
110:     LOCAL length%                                          'number of bytes read
112:
113:     out.string$ = STRING$(%PGA.max.string, " ")            'create a string to put data in
114:     CALL PGA.error.read(length%, out.string$)              'get data from PGA
115:
116:     IF (length%=0) THEN                                    'return null string if no bytes read
117:         out.string$ = ""
118:     ELSE                                                   'else return a string of "length" bytes
119:         out.string$ = LEFT$(out.string$, length%)
120:     END IF
121: END SUB
122:
123:
124: SUB PGA.print.error( out.string$ )
125:
126:     CALL PGA.transmit("DI,1 ")                             'switch to emulator screen
127:     CALL print.error( out.string$ )                        'print message
128:
129: END SUB
130:
131:
132: SUB PGA.recieve( out.string$ )
133:     LOCAL length%                                          'number of bytes read
134:
135:     out.string$ = STRING$(%PGA.max.string, " ")            'create a string to put data in
136:     CALL PGA.read(length%, out.string$)                    'get data from PGA
137:
138:     IF (length%=0) THEN                                    'return null string if no bytes read
139:         out.string$ = ""
140:     ELSE                                                   'else return a string of "length" bytes
141:         out.string$ = LEFT$(out.string$, length%)
142:     END IF
143: END SUB
144:
145:
146: SUB PGA.warm.restart
147:     DEF SEG = %PGA.base                          'set segment
148:     POKE %PGA.warm.restart.flag, %PGA.set        'set warm restart flag
149:     DEF SEG                                      'return to BASIC segment
150: END SUB
151:
152:
153:
154: '
155: '                        INLINE SUBROUTINES
156: '
157:
158: SUB PGA.transmit INLINE  '( in.string$ )
159:         $INLINE "\INCLUDE\PGAtran.bin"
160: END SUB
161:
162:
163: SUB PGA.read INLINE  '( length%, out.string$ )
164:         $INLINE "\INCLUDE\PGArecv.bin"
```

```
END SUB


SUB PGA.error.read INLINE  '( length%, out.string$ )
        $INLINE "\INCLUDE\PGAerrcv.bin"
END SUB
```

```
 1: '┌──────────────────────────────────────────────────────────────────────┐
 2: '
 3: '      FILNAME:                pgagraph.BAS
 4: '      CREATED:                -1/8706.12
 5: '      LAST MODIFIED:          -1/8810.22
 6: '      PROGRAMMER:             Christopher Voltz - UDRI
 7: '      TARGET:                 IBM PC w/PGA
 8: '      LANGUAGE:               Turbo BASIC
 9: '
10: '      PURPOSE:
11: '            This program reads in a data file consisting of
12: '      bandwith number and 200 X, Y coordinate pairs which
13: '      represent vectors which are joined end to end.  The
14: '      data is scaled to take into account the perspective of
15: '      the graphic mode used (640x350).  The data is then
16: '      displayed on a standard cartesian coordinate system.
17: '      The user must specify the path name of the target file.
18: '      An extension of .DAT is assumed if none is specified.
19: '      The user can then manipulate the graph using the keys
20: '      described in the initial display page.  When the user
21: '      presses the ESC key, the program terminates.
22: '
23: '└──────────────────────────────────────────────────────────────────────┘
24:
25:
26: '┌──────────────────────────────────────────────────────────────────────┐
27: '                              INITIALIZATION
28: '└──────────────────────────────────────────────────────────────────────┘
29:
30:         '**** define toolbox constants
31: %error.color = 12
32: %escape.key  = 27
33: %false       =  0
34: %indent      =  5
35: %text.color  = 15
36: %menu.color  = 15
37: %true        = -1
38:
39:         '**** define string constants
40: null$        = CHR$(0)                  'null character string
41: del$         = null$ + CHR$(83)         'scan codes returned by delete key
42: down.arrow$  = null$ + CHR$(80)         'down arrow key
43: fun.1$       = null$ + CHR$(59)         'function key 1
44: fun.2$       = null$ + CHR$(60)         'function key 2
45: fun.3$       = null$ + CHR$(61)         'function key 3
46: fun.4$       = null$ + CHR$(62)         'function key 4
47: fun.5$       = null$ + CHR$(63)         'function key 5
48: fun.6$       = null$ + CHR$(64)         'function key 6
49: fun.7$       = null$ + CHR$(65)         'function key 7
50: fun.8$       = null$ + CHR$(66)         'function key 8
51: fun.9$       = null$ + CHR$(67)         'function key 9
52: fun.10$      = null$ + CHR$(68)         'function key 10
53: fun.11$      = null$ + CHR$(84)         'fn1 + shift
54: fun.12$      = null$ + CHR$(85)         'fn2 + shift
```

```
 55: fun.13$      = null$ + CHR$(86)          'fn3 + shift
 57: fun.14$      = null$ + CHR$(87)          'fn4 + shift
 58: fun.15$      = null$ + CHR$(88)          'fn5 + shift
 59: fun.16$      = null$ + CHR$(89)          'fn6 + shift
 60: fun.17$      = null$ + CHR$(90)          'fn7 + shift
 61: fun.19$      = null$ + CHR$(92)          'fn9 + shift
 62: fun.20$      = null$ + CHR$(93)          'fn10 + shift
 63: ins$         = null$ + CHR$(82)          'insert key
 64: left.arrow$  = null$ + CHR$(75)          'left arrow key
 65: minus$       = CHR$(45)                  'minus key
 66: plus$        = CHR$(43)                  'plus key
 67: prt.sc$      = CHR$(42)                  'print screen key
 68: right.arrow$ = null$ + CHR$(77)          'right arrow key
 69: up.arrow$    = null$ + CHR$(72)          'up arrow key
 70: terminate$   = CHR$(27)                  'escape key
 71:
 72: fix.file.name$ = "screens\fixate.img"    'file name of fixation point
 73:
 74:         '**** define program constants and initialize variables
 75: PI           = 4 * ATN(1)
 76: lambda.1!    = 0                         'rotate image by lambda radians
 77: lambda.2!    = 0                         'rotate fundamental by lambda radians
 78: %block.size = 30000                      'number of bytes in an I/O block
 79:
 80: aspect.ratio = 4 / 3                     'standard screen aspect ratio
 81: %screen.x    = 320                       'half the horizontal size in pixels
 82: %screen.y    = 240                       'half the vertical size in pixels
 83: pixel.mm!    = 0.36                      'number of pixels per mm
 84: scale        = %screen.x / %screen.y / aspect.ratio     '2.55
 85:              ' scale factor for y-axis to take into account the
 86:              ' perspective ratio of the high resolution mode
 87:
 88: fill%     = %false                      'boolean: if image is filled or not
 89: x.offset% = 0  '110                     'offset in x direction
 90: y.offset% = 0                           'offset in y direction
 91: rot%      = 0                           'rotation in z direction in degrees
 92: inc%      = 10                          'offset step size
 93: demag.inc = 0.5                         'demagnification factor step size
 94: demag     = 13'39.5                     'demagnification factor of screen
 95:
 96: amp.mult = 0.0                          'amplitude multiplier
 97: amp.step = 0.07                         'amplitude step size (alpha increment)
 98: amp.inc  = 0                            'amplitude increment step size
 99:                                         '(calculated later)
100:
101: box.size.x! = 1800                      'size of digitizing box in virtual pixels
102: box.size.y! = 2150                      'center of box is at (0,0) in the
103:                                         'virtual coordinate system
104: %radius  = 15                           'size of "hole" in noise screen
105: density& = 77000                       'number of times to plot points on screen
106:
107: plot.size! = 2.54 / 5                   'size of plot (conversion factor / #CM)
108:                                         '{largest size = 35 cm}
109: %plot.x    = 2348 '2280                 'size of plotter in X direction
```

```
110: %plot.y    = 1761 '1711                    'size of plotter in Y direction
112: %y.offset  = 50
113:
114:         '**** define arrays to hold coefficients and coordinates
115: DIM x%(200), y%(200), I(300), IY(300)
116: DIM AN(61), BN(61), CN(61), DN(61), T(300)
117:
118:
119: $include "\include\pga.inc"
120: $include "\include\toolbox.inc"
121:
122: '
123: '                              MAIN PROGRAM
124: '
125:
126: ON ERROR GOTO error.handler
127:
128: KEY OFF
129:
130: CALL show.help.screen
131:
132: CALL get.magnitude.data(number.of.links%, t(), an(), bn(), cn(), dn())
133:
134: CLS
135: INPUT "ENTER HARMONIC FREQUENCY";H
136: CALL calculate.coordinates(alpha!, amp.inc, amp.mult, amp.step, pi, h, _
137:                            number.of.links%, t(), i(), iy(), an(), bn(), _
138:                            cn(), dn())
139:
140: restart:
141:
142: CALL PGA.warm.restart
143: DELAY 0.5
144: CALL PGA.error.enable
145: CALL PGA.transmit("CA LI,5 L,0,0,0,0 L,255,15,15,15 L,5,15,0,0 ")
146: CALL PGA.transmit("L,1,15,15,15 L,2,15,15,15 L,3,15,15,15 L,6,15,15,15 ")
147: CALL PGA.transmit("PF,0 MK,255 DI,0 C,255 ")
148:
149: DO
150:         CALL setup.screen(x.offset%, demag, y.offset%, scale)
151:
152:                         'display image info
153:         CALL PGA.transmit("M,"+STR$((-%screen.x-x.offset%)*demag)+","+
154:                           STR$((-%screen.y-y.offset%)*demag*scale+40)+",","+_
155:                           " T '("+STR$(INT(x.offset%*pixel.mm!))+", "+_
156:                           STR$(INT(y.offset%*pixel.mm!))+")"+_
157:                           " DEMAG="+STR$(CSNG(demag))+_
158:                           " H="+STR$(h)+
159:                           " AMP="+LEFT$(STR$(CSNG(amp.mult*alpha!)),6)+";"+_
160:                           " PHI.1="+LEFT$(STR$(CSNG(lambda.1!*180/PI)),6)+_
161:                           " PHI.2="+LEFT$(STR$(CSNG(lambda.2!*180/PI)),6)+",";"+_
162:                           " ROT= "+LEFT$(STR$(CSNG(rot!*180/PI)),6)+"' ")
163:
164:                         'draw the image

FILE=pgagraph.bas    Wed Jun 14 16:23:01 1989    PAGE=3
```

```
165:        IF fill% THEN
167:            CALL PGA.transmit("C,5 ")
168:            CALL PGA.transmit("M,"+STR$(i(1))+STR$(iy(1))+" ")
169:            FOR index% = 2 TO number.of.links% + 1 '200
170:                CALL PGA.transmit("D "+STR$(i(index%))+","+STR$(iy(index%))+" ")
171:            NEXT
172:            CALL PGA.transmit("C,255 M,0,0 A ")
173:        END IF
174:        CALL PGA.transmit("C,255 M,"+STR$(i(1))+STR$(iy(1))+" ")
175:        FOR index% = 2 TO number.of.links% + 1 '200
176:            CALL PGA.transmit("D "+STR$(i(index%))+","+STR$(iy(index%))+" ")
177:        NEXT
178:
179:                        'draw image grabber rectangle and measuring points
180:        CALL PGA.transmit("M,"+STR$((-x.offset%)*demag)+","+STR$((-y.offset%)*demag)+" PT ")
181:        CALL PGA.transmit("M,0,0 PT ")
182:        CALL PGA.transmit("C,5 ")
183:        CALL PGA.transmit("M,"+STR$(-box.size.x!/2)+","+STR$(-box.size.y!/2)+" "+_
184:                          "RR,"+STR$(box.size.x!)+","+STR$(box.size.y!)+" ")
185:        CALL PGA.transmit("C,255 ")
186:
187:                        'used to create fixation point
188: '      CALL PGA.transmit("PF,1 M,0,"+STR$(4*demag)+" RR,"+STR$(demag)+","+_
189: '                        STR$(-7*demag)+" M,"+STR$(-3*demag)+",0 RR,"+_
190: '                        STR$(7*demag)+","+STR$(demag)+" PF,0 ")
191:
192:        WHILE NOT INSTAT                        'wait until a key is pressed
193:        WEND
194:
195:        keypressed$ = INKEY$
196:        SELECT CASE keypressed$
197:            CASE minus$
198:                INCR demag, demag.inc   'increase demagnification
199:                IF demag = 0 THEN _
200:                   demag = demag.inc    'make sure demag is never zero
201:            CASE plus$
202:                DECR demag, demag.inc   'increase magnification
203:                IF demag = 0 THEN
204:                   demag = -demag.inc   'make sure demag is never zero
205:            CASE del$
206:                DECR scale, 0.01        'decrease scale
207:            CASE ins$
208:                INCR scale, 0.01        'increase scale
209:            CASE left.arrow$
210:                DECR x.offset%, inc%    'move screen left
211:            CASE right.arrow$
212:                INCR x.offset%, inc%    'move screen right
213:            CASE up.arrow$
214:                INCR y.offset%, inc%    'move screen up
215:            CASE down.arrow$
216:                DECR y.offset%, inc%    'move screen down
217:            CASE prt.sc$
218:                LPRINT " AMP.MULT= "; csng(amp.mult);
219:                LPRINT " AMPLITUDE= "; amp.mult * alpha!

FILE=pgagraph.bas    Wed Jun 14 16:23:01 1989    PAGE=4
```

```
220:                LPRINT USING "    INC= #####"; inc%;
222:                LPRINT USING "    DEMAG= #####.###"; demag
223:                LPRINT " X.OFFSET= "; INT(x.offset%*pixel.mm!);
224:                LPRINT " Y.OFFSET= "; INT(y.offset%*pixel.mm!)
225:                LPRINT " ROT= ";ROT!*180/PI
226:                LPRINT USING " BOX.SIZE.X= ######.###"; box.size.x!
227:                LPRINT USING " BOX.SIZE.Y= ######.###"; box.size.y!
228:            CASE fun.1$
229:                CALL PGA.transmit("DI,1 ")
230:                CALL show.help.screen   'show help screen
231:                CALL PGA.transmit("DI,0 ")
232:            CASE fun.2$
233:                DECR box.size.y!, demag 'decrease digitized Y image size in
234:            CASE fun.3$
235:                INCR box.size.y!, demag 'increase digitized Y image size
236:            CASE fun.4$                 'select new harmonic
237:                CALL PGA.transmit("DI,1 ")
238:                SCREEN 0
239:                CLS
240:                INPUT "NEW HARMONIC FREQUENCY";H
241:                amp.mult = 0
242:                CALL calculate.coordinates(alpha!, amp.inc, amp.mult, amp.step, _
243:                                           pi, h, number.of.links%, t(), i(), _
244:                                           iy(), an(), bn(), cn(), dn())
245:                CALL PGA.transmit("DI,0 ")
246:            CASE fun.5$                       'decrease amplitude
247:                DECR amp.mult, amp.inc
248:                CALL calculate.coordinates(alpha!, amp.inc, amp.mult, amp.step, _
249:                                           pi, h, number.of.links%, t(), i(), _
250:                                           iy(), an(), bn(), cn(), dn())
251:            CASE fun.6$                       'increase amplitude
252:                INCR amp.mult, amp.inc
253:                CALL calculate.coordinates(alpha!, amp.inc, amp.mult, amp.step, _
254:                                           pi, h, number.of.links%, t(), i(), _
255:                                           iy(), an(), bn(), cn(), dn())
256:            CASE fun.7$
257:                CALL plot.image(i(), iy(), x.offset%, demag, y.offset%, scale)
258:            CASE fun.8$
259:                fill% = NOT fill%
260:            CASE fun.9$
261:                CALL read.image
262:            CASE fun.10$
263:                CALL save.image
264:            CASE fun.11$
265:                CALL PGA.transmit("DI,1 ")
266:                SCREEN 0
267:                CLS
268:                INPUT "NEW PHASE FOR FUNDAMENTAL (degrees) ";lambda.2!
269:                INPUT "NEW PHASE FOR NON-FUNDAMENTAL (degrees)";lambda.1!
270:                lambda.1! = (lambda.1! * PI/180)
271:                lambda.2! = (lambda.2! * PI/180)
272:                CALL calculate.coordinates(alpha!, amp.inc, amp.mult, amp.step, _
273:                                           pi, h, number.of.links%, t(), i(), _
274:                                           iy(), an(), bn(), cn(), dn())

FILE=pgagraph.bas    Wed Jun 14 16:23:01 1989    PAGE=5
```

```
275:                CALL PGA.transmit("DI,0 ")
277:            CASE fun.12$
278:                DECR box.size.x!, demag 'decrease digitized X image size in
279:            CASE fun.13$
280:                INCR box.size.x!, demag 'increase digitized X image size
281:            CASE fun.14$
282:                CALL read.screen
283:            CASE fun.15$
284:                CALL PGA.transmit("DI,1 ")
285:                SCREEN 0
286:                CLS
287:                INPUT "NEW ROTATION ANGLE (degrees) ";rot!
288:                rot! = (rot! * PI/180)
289:                CALL calculate.coordinates(alpha!, amp.inc, amp.mult, amp.step, _
290:                                           pi, h, number.of.links%, t(), i(), _
291:                                           iy(), an(), bn(), cn(), dn())
292:                CALL PGA.transmit("DI,0 ")
293:            CASE fun.16$
294:                CALL make.noise.screen
295:            CASE fun.17$
296:                CALL PGA.transmit("DI,1 ")
297:                SCREEN 0
298:                CLS
299:                INPUT "Enter new amplitude level: ";amp.mult
300:                amp.mult = amp.mult / alpha!
301:                CALL calculate.coordinates(alpha!, amp.inc, amp.mult, amp.step, _
302:                                           pi, h, number.of.links%, t(), i(), _
303:                                           iy(), an(), bn(), cn(), dn())
304:                CALL PGA.transmit("DI,0 ")
305:            CASE fun.19$
306:                CALL read.2.images
307:            CASE fun.20$
308:                CALL PGA.transmit("DI,1 ")
309:                SCREEN 0
310:                CLS
311:                INPUT "ENTER COMMAND TO SHELL (<CR> TO GO TO DOS): ";keypressed$
312:                SHELL keypressed$
313:                CALL PGA.transmit("DI,0 ")
314:        END SELECT
315:
316: LOOP UNTIL keypressed$ = terminate$             'exit when escape pressed
317:
318: CALL PGA.transmit("DI,1 ")
319: SCREEN 0                                       'text screen
320: END
321:
322: error.handler:
323:     CALL PGA.error.recieve(recieve$)            'if an error occurs, return to
324:     CALL PGA.recieve(recieve$)                  'CGA emulation mode and print
325:     CALL PGA.transmit("CA DI,1 ")               'an error message.
326:     CALL print.error ("ERROR "+STR$(ERR)+": "+FNget.error.message$+" at "+STR$(ERADR))
327:     CALL confirm(recieve$)
328:     CALL PGA.transmit("DI,0 ")
329:     RESUME restart
```

```
330:
332:
333:
334:
335: '
336: '                          SUBROUTINES
337: '
338:
339: SUB calculate.coordinates(alpha!, amp.inc, amp.mult, amp.step, pi, h,
340:                           number.of.links%, t(1), i(1), iy(1), an(1), Bn(1), _
341:                           cn(1), dn(1))
342:     LOCAL s%, vr, x2, y2, theta!, mag!
343:     SHARED lambda.1!, lambda.2!, rot!
344:
345:         FOR S%=1 TO number.of.links% + 1 '299
346:             VR=(2*PI*H*S%)/T(number.of.links%) + lambda.1!
347:             X2=amp.mult * (AN(H)*COS(VR) + BN(H)*SIN(VR))
348:             Y2=amp.mult * (CN(H)*COS(VR) + DN(H)*SIN(VR))
349:             IF H<>1 THEN                'add first harmonic if necessary
350:                 VR=(2*PI*S%)/T(number.of.links%) + lambda.2!
351:                 X2=AN(1)*COS(VR) + BN(1)*SIN(VR) + X2
352:                 Y2=CN(1)*COS(VR) + DN(1)*SIN(VR) + Y2
353:             END IF
354:             theta! = rot!+ATN(Y2/X2)-PI*(X2<0)-2*PI*(X2>0)
355:             mag! = SQR(X2^2+Y2^2)
356:             I(S%)=INT(100*mag!*COS(theta!))
357:             IY(S%)=INT(100*mag!*SIN(theta!))
358:         NEXT
359:
360:         'alpha = sqr( (magnitude of x) ^2  +  (magnitude of y) ^2 )
361:         '    magnitude = sqr( real.part^2 + imaginary.part^2 )
362:         alpha! = sqr(an(H)^2 + bn(h)^2 + cn(h)^2 + dn(h)^2)
363:         amp.inc = amp.step / alpha!
364: END SUB
365:
366:
367:
368:
369: SUB get.magnitude.data(number.of.links%, t(1), an(1), bn(1), cn(1), dn(1))
370:     LOCAL file.name$, index%
371:
372:         COLOR %text.color
373:         INPUT "Filename of data file: (.DAT) ",file.name$
374:
375:         OPEN file.name$+".DAT" FOR INPUT AS #1
376:             INPUT #1, number.of.links%
377:             FOR index% = 1 to number.of.links%
378:                INPUT #1, t(index%)
379:             NEXT
380:             FOR index% = 1 to 60
381:                INPUT #1, AN(index%), BN(index%), CN(index%), DN(index%)
382:             NEXT
383:         CLOSE #1
384: END SUB
```

FILE=pgagraph.bas Wed Jun 14 16:23:01 1989 PAGE=7

```
385:
387:
388:
389:
390: '
391: '  MAKE.NOISE.SCREEN
392: '
393: '
394: '          This module generates a random set of coordinates and
395: '       sets them (lighting them on the screen).  This is done until the
396: '       density& points have been plotted.  Note: a small circle
397: '       centered in the center of the screen is left with nothing in
398: '       it.  This is to provide room for the fixation point.
399: '
400: '
401:
402: SUB make.noise.screen
403:
404:     LOCAL index&         'loop control
405:     LOCAL tmp.string$    'temporary string
406:     LOCAL x%             'chosen x coordinate
407:     LOCAL y%             'chosen y coordindate
408:
409:     SHARED density&      'number of points to plot on screen
410:     SHARED terminate$    'code returned by <ESC>
411:
412:     RANDOMIZE TIMER
413:     CALL PGA.transmit("MK,255 CLS,0 C,255 WI,"+STR$(-%screen.x)+","+ _
414:                       STR$(%screen.x)+","+STR$(-%screen.y)+","+_
415:                       STR$(%screen.y)+" ")
416:     FOR index& = 1 TO density&
417:         x& = INT(2*%screen.x*RND) - %screen.x
418:         y& = INT(2*%screen.y*RND) - %screen.y
419:         IF ((x&*x& + y&*y&) > (%radius*%radius)) THEN
420:              CALL PGA.transmit("M,"+STR$(x&)+","+STR$(y&)+" PT ")
421:              CALL PGA.error.recieve(tmp.string$)
422:              IF tmp.string$<>"" THEN
423:                 CALL PGA.print.error("PGA ERROR: #="+STR$(ASC(tmp.string$)))
424:                 CALL PGA.transmit("CX ")
425:              END IF
426:         END IF
427:         IF (INKEY$ = terminate$) THEN EXIT FOR
428:      NEXT
429:
430:      CALL PGA.transmit("DI,1 ")
431:      SCREEN 0
432:      CLS
433:      INPUT "Filename to save screen under? (.SCR): ";tmp.string$
434:      tmp.string$ = "screens\" + tmp.string$
435:      SHELL "ERASE "+tmp.string$+".SCR"
436:      OPEN tmp.string$+".SCR" FOR BINARY AS #1
437:      CALL PGA.transmit("CX ")
438:      CALL PGA.recieve(tmp.string$)              'clear buffer
439:
```

```
440:      FOR y% = 0 TO 2*%screen.y-1
442:         CALL PGA.transmit(CHR$(&HD8)+                              'IR
443:                           CHR$(y% MOD 256) + _                     'lowline
444:                           CHR$(y% \ 256) + _                       'highline
445:                           CHR$(0 MOD 256) + _                      'lowX1 (start)
446:                           CHR$(0 \ 256) + _                        'highX1
447:                           CHR$((2*%screen.x -1) MOD 256) + _       'lowX2 (end)
448:                           CHR$((2*%screen.x -1) \ 256) )           'highX2
449:         CALL PGA.recieve(tmp.string$)
450:         WHILE (tmp.string$<>"")
451:             PUT$ #1, tmp.string$
452:             CALL PGA.error.recieve(tmp.string$)
453:             IF tmp.string$<>"" THEN
454:                 CALL print.error("PGA ERROR: #="+STR$(ASC(tmp.string$)))
455:                 CALL PGA.transmit("CX ")
456:             END IF
457:             DELAY 0.05                          'give PGA time to write data
458:             CALL PGA.recieve(tmp.string$)
459:         WEND
460:     NEXT
461:     CLOSE #1
462:
463:     CALL PGA.transmit("CA DI,0 ")
464:     BEEP
465:
466: END SUB
467:
468:
469:
470:
471: SUB plot.image(i(1), iy(1), x.offset%, demag, y.offset%, scale)
472:     LOCAL       index%
473:     LOCAL       x.convert!, y.convert!, x.fact!, y.fact!
474:     SHARED      plot.size!, number.of.links%
475:
476:         ' initialize plotter
477:     OPEN "COM1:2400,O,7,2,cd,ds,cs" AS #1
478:     PRINT #1, ";:EH ECM H A ";
479:     DELAY 0.5
480:
481:         ' calculate scaling factors
482:     x.fact! = demag*(%screen.x-x.offset%)*plot.size!
483:     y.fact! = scale*demag*(%screen.y-y.offset%)*plot.size!
484:     x.convert! = %plot.x / (2*x.fact!)
485:     y.convert! = %plot.y / (2*y.fact!)
486:
487:     PRINT #1, "P1 "
488:     DELAY 4.0
489:
490:     PRINT #1, STR$(CINT((i(1)+x.fact!)*x.convert!)); ",";
491:     PRINT #1, STR$(CINT((iy(1)+y.fact!)*y.convert!)+%y.offset);" D ";
492:     DELAY 0.5
493:     FOR index% = 2 TO number.of.links% + 1
494:         PRINT #1, STR$(CINT((i(index%)+x.fact!)*x.convert!));",";

FILE=pgagraph.bas    Wed Jun 14 16:23:01 1989    PAGE=9
```

```
495:         PRINT #1, STR$(CINT((iy(index%)+y.fact!)*y.convert!)+%y.offset)
497:     NEXT
498:
499:     PRINT #1, "U P0 ";
500:     CLOSE #1
501:
502: END SUB
503:
504:
505:
506:
507: '
508: '     READ.IMAGE
509: '
510: '
511: '          This module loads in an image (in hex format) from the file
512: '       specified.
513: '
514: '
515:
516: SUB read.image
517:
518:         ' **** VARIABLE DECLARATIONS
519:     LOCAL file.size&     'file size in bytes
520:     LOCAL tmp.string$    'temporary string
521:
522:         ' *** get filename
523:     CALL PGA.transmit("DI,1 CLS,0 ")
524:     SCREEN 0
525:     CLS
526:     INPUT "Filename to read image from? (.IMG) ", tmp.string$
527:
528:         ' *** open file and get size
529:     tmp.string$ = "screens\" + tmp.string$
530:     OPEN tmp.string$ + ".IMG" FOR BINARY AS #1
531:     file.size& = LOF(1)                                    'get file size
532:     IF file.size& = 0 THEN                                 'if file does not exist
533:        CLOSE #1                                            'notify user
534:        SHELL "ERASE "+tmp.string$+".IMG"
535:        CALL print.error("ERROR: "+tmp.string$+" does not exist.")
536:        EXIT SUB
537:     END IF
538:
539:         ' *** load in image and transmit to PGA
540:     SEEK #1, 0                                             'goto beginning of file
541:
542:     CALL PGA.error.recieve(tmp.string$)                    'clear error buffer
543:     IF tmp.string$ <> "" THEN
544:        CALL print.error("PGA ERROR: "+tmp.string$)
545:     END IF
546:     CALL PGA.transmit("CX ")                               'switch to nex mode
547:
548:     WHILE NOT EOF(1)
549:         GET$ #1, %block.size, tmp.string$                  'read in a bufferfull
```

```
550:         CALL PGA.transmit(tmp.string$)                                  'send to the PGA
552:         CALL PGA.error.recieve(tmp.string$)                              'check for errors
553:         IF tmp.string$ <> "" THEN
554:             CALL print.error("PGA ERROR NUMBER: "+STR$(ASC(tmp.string$)))
555:         END IF
556:     WEND
557:
558:     CLOSE #1                                                            'close the data file
559:
560:     CALL PGA.error.recieve(tmp.string$)                                'check for errors
561:     IF tmp.string$ <> "" THEN
562:         CALL print.error("PGA ERROR NUMBER: "+STR$(ASC(tmp.string$)))
563:     END IF
564:
565:     CALL PGA.transmit("CA DI,0 ")                                       'return to ASCII mode
566:
567:     BEEP
568:     WHILE INKEY$="": WEND                                               'display until user presses a key
569:
570: END SUB
571:
572:
573:
574:
575: '
576: '    READ.SCREEN
577: '
578: '
579: '        This module loads in a screen (in hex format) from the file
580: '    specified.
581: '
582: '
583:
584: SUB read.screen
585:
586:         ' **** VARIABLE DECLARATIONS
587:     LOCAL file.size&    'file size in bytes
588:     LOCAL tmp.string$   'temporary string
589:
590:         ' *** get filename
591:     CALL PGA.transmit("DI,1 CLS,0 ")
592:     SCREEN 0
593:     CLS
594:     INPUT "Filename to read screen from? (.SCR) ", tmp.string$
595:
596:         ' *** open file and get size
597:     tmp.string$ = "screens\" + tmp.string$
598:     OPEN tmp.string$ + ".SCR" FOR BINARY AS #1
599:     file.size& = LOF(1)                                                 'get file size
600:     IF file.size& = 0 THEN                                              'if file does not exist
601:        CLOSE #1                                                         'notify user
602:        SHELL "ERASE "+tmp.string$+".IMG"
603:        CALL print.error("ERROR: "+tmp.string$+" does not exist.")
604:        EXIT SUB
```

```
605:     END IF
607:
608:         ' *** load in image and transmit to PGA
609:     SEEK #1, 0                                                    'goto beginning of file
610:
611:     CALL PGA.error.recieve(tmp.string$)                           'clear error buffer
612:     IF tmp.string$ <> "" THEN
613:        CALL print.error("PGA ERROR: "+tmp.string$)
614:     END IF
615:     CALL PGA.transmit("CX ")                                      'switch to hex mode
616:
617:     WHILE NOT EOF(1)
618:         GET$ #1, %block.size, tmp.string$                         'read in a bufferfull
619:         CALL PGA.transmit(tmp.string$)                            'send to the PGA
620:         CALL PGA.error.recieve(tmp.string$)                        'check for errors
621:         IF tmp.string$ <> "" THEN
622:             CALL print.error("PGA ERROR NUMBER: "+STR$(ASC(tmp.string$)))
623:         END IF
624:     WEND
625:
626:     CLOSE #1                                                      'close the data file
627:
628:     CALL PGA.error.recieve(tmp.string$)                          'check for errors
629:     IF tmp.string$ <> "" THEN
630:         CALL print.error("PGA ERROR NUMBER: "+STR$(ASC(tmp.string$)))
631:     END IF
632:
633:     CALL PGA.transmit("CA DI,0 ")                                 'return to ASCII mode
634:
635:     BEEP
636:     WHILE INKEY$="": WEND                                         'display until user presses a key
637:
638: END SUB
639:
640:
641:
642:
643: '
644: '     READ.2.IMAGES
645: '
646: '
647: '         This module loads in two images (in hex format) from the
648: '      files specified.  In addition, the fixation point is loaded.
649: '
650: '
651:
652: SUB read.2.images
653:
654:         ' **** VARIABLE DECLARATIONS
655:     LOCAL file.size&    'file size in bytes
656:     LOCAL tmp.string$   'temporary string
657:
658:         ' **** GLOBAL DECLARATIONS
659:     SHARED fix.file.name$       'file name of fixation point
```

```
660:
662:        ' *** get filename
663:     CALL PGA.transmit("DI,1 CLS,0 ")
664:     SCREEN 0
665:     CLS
666:     INPUT "Filename to read first image from? (.IMG) ", tmp.string$
667:
668:        ' *** open file and get size
669:     tmp.string$ = "screens\" + tmp.string$
670:     OPEN tmp.string$ + ".IMG" FOR BINARY AS #1
671:     file.size& = LOF(1)                                          'get file size
672:     IF file.size& = 0 THEN                                       'if file does not exist
673:        CLOSE #1                                                  'notify user
674:        SHELL "ERASE "+tmp.string$+".IMG"
675:        CALL print.error("ERROR: "+tmp.string$+" does not exist.")
676:        EXIT SUB
677:     END IF
678:
679:        ' *** load in image and transmit to PGA
680:     SEEK #1, 0                                                   'goto beginning of file
681:
682:     CALL PGA.error.recieve(tmp.string$)                          'clear error buffer
683:     IF tmp.string$ <> "" THEN
684:        CALL print.error("PGA ERROR: "+tmp.string$)
685:     END IF
686:     CALL PGA.transmit("CX ")                                     'switch to hex mode
687:
688:     WHILE NOT EOF(1)
689:        GET$ #1, %block.size, tmp.string$                         'read in a bufferfull
690:        CALL PGA.transmit(tmp.string$)                            'send to the PGA
691:        CALL PGA.error.recieve(tmp.string$)                        'check for errors
692:        IF tmp.string$ <> "" THEN
693:           CALL print.error("PGA ERROR NUMBER: "+STR$(ASC(tmp.string$)))
694:        END IF
695:     WEND
696:
697:     CLOSE #1                                                     'close the data file
698:
699:     CALL PGA.error.recieve(tmp.string$)                         'check for errors
700:     IF tmp.string$ <> "" THEN
701:        CALL print.error("PGA ERROR NUMBER: "+STR$(ASC(tmp.string$)))
702:     END IF
703:
704:     INPUT "Filename to read second image from? (.IMG) ", tmp.string$
705:
706:        ' *** open file and get size
707:     tmp.string$ = "screens\" + tmp.string$
708:     OPEN tmp.string$ + ".IMG" FOR BINARY AS #1
709:     file.size& = LOF(1)                                          'get file size
710:     IF file.size& = 0 THEN                                       'if file does not exist
711:        CLOSE #1                                                  'notify user
712:        SHELL "ERASE "+tmp.string$+".IMG"
713:        CALL print.error("ERROR: "+tmp.string$+" does not exist.")
714:        EXIT SUB
```

```
715:     END IF
717:
718:         ' *** load in image and transmit to PGA
719:     SEEK #1, 0                                                    'goto beginning of file
720:
721:     CALL PGA.error.recieve(tmp.string$)                           'clear error buffer
722:     IF tmp.string$ <> "" THEN
723:        CALL print.error("PGA ERROR: "+tmp.string$)
724:     END IF
725:     CALL PGA.transmit("CX ")                                      'switch to hex mode
726:
727:     WHILE NOT EOF(1)
728:         GET$ #1, %block.size, tmp.string$                         'read in a bufferfull
729:         CALL PGA.transmit(tmp.string$)                            'send to the PGA
730:         CALL PGA.error.recieve(tmp.string$)                        'check for errors
731:         IF tmp.string$ <> "" THEN
732:             CALL print.error("PGA ERROR NUMBER: "+STR$(ASC(tmp.string$)))
733:         END IF
734:     WEND
735:
736:     CLOSE #1                                                      'close the data file
737:
738:     CALL PGA.error.recieve(tmp.string$)                          'check for errors
739:     IF tmp.string$ <> "" THEN
740:         CALL print.error("PGA ERROR NUMBER: "+STR$(ASC(tmp.string$)))
741:     END IF
742:
743:         ' *** open file and get size
744:     PRINT "LOADING FIXATION POINT."
745:     OPEN fix.file.name$ FOR BINARY AS #1
746:     file.size& = LOF(1)                                           'get file size
747:     IF file.size& = 0 THEN                                        'if file does not exist
748:        CLOSE #1                                                   'notify user
749:        SHELL "ERASE "+fix.file.name$
750:        CALL print.error("ERROR: "+fix.file.name$+" does not exist.")
751:        EXIT SUB
752:     END IF
753:
754:         ' *** load in image and transmit to PGA
755:     SEEK #1, 0                                                    'goto beginning of file
756:
757:     CALL PGA.error.recieve(tmp.string$)                           'clear error buffer
758:     IF tmp.string$ <> "" THEN
759:        CALL print.error("PGA ERROR: "+tmp.string$)
760:     END IF
761:     CALL PGA.transmit("CX ")                                      'switch to hex mode
762:
763:     WHILE NOT EOF(1)
764:         GET$ #1, %block.size, tmp.string$                         'read in a bufferfull
765:         CALL PGA.transmit(tmp.string$)                            'send to the PGA
766:         CALL PGA.error.recieve(tmp.string$)                        'check for errors
767:         IF tmp.string$ <> "" THEN
768:             CALL print.error("PGA ERROR NUMBER: "+STR$(ASC(tmp.string$)))
769:         END IF

FILE=pgagraph.bas    Wed Jun 14 16:23:01 1989    PAGE=14
```

```
770:     WEND
772:
773:     CLOSE #1                                                    'close the data file
774:
775:     CALL PGA.error.recieve(tmp.string$)                        'check for errors
776:     IF tmp.string$ <> "" THEN
777:         CALL print.error("PGA ERROR NUMBER: "+STR$(ASC(tmp.string$)))
778:     END IF
779:
780:     CALL PGA.transmit("CA DI,0 ")                               'return to ASCII mode
781:
782:     BEEP
783:     WHILE INKEY$="": WEND                                       'display until user presses a key
784:
785: END SUB
786:
787:
788:
789:
790: SUB save.image
791:    LOCAL tmp.string$, y.start%, y.end%, x.start%, x.end%
792:    SHARED box.size.x!, box.size.y!, y.offset%, x.offset%, demag, scale
793:
794:    CALL PGA.transmit("DI,1 ")
795:    SCREEN 0
796:    CLS
797:
798:    INPUT "Filename to save image under? (.IMG) ", tmp.string$
799:    tmp.string$ = "screens\" + tmp.string$
800:    SHELL "ERASE "+tmp.string$+".IMG"
801:    OPEN tmp.string$+".IMG" FOR BINARY AS #1
802:    CALL PGA.transmit("CX ")
803:    CALL PGA.recieve(tmp.string$)                  'clear buffer
804:
805:    x.start% = (-box.size.x!/2+(%screen.x+x.offset%)*demag)*((2*%screen.x-1)/(2*demag*%screen.x))+1
806:    x.end%   = ( box.size.x!/2+(%screen.x+x.offset%)*demag)*((2*%screen.x-1)/(2*demag*%screen.x))-1
807:    y.start% = (-box.size.y!/2+(%screen.y+y.offset%)*demag*scale)*((2*%screen.y-1)/(2*demag*scale*%screen.y))+1
808:    y.end%   = ( box.size.y!/2+(%screen.y+y.offset%)*demag*scale)*((2*%screen.y-1)/(2*demag*scale*%screen.y))-1
809:    IF (x.start%<0) OR (x.end%>%screen.x*2-1) OR
810:       (y.start%<0) OR (y.end%>%screen.y*2-1) THEN
811:         CALL print.error("ERROR: digitizing box is too large.")
812:         CLOSE #1
813:         CALL PGA.transmit("DI,0 ")
814:         EXIT SUB
815:    END IF
816:
817:    FOR y.coord% = y.start% TO y.end%
818:         CALL PGA.transmit(CHR$(&HD8)+                          'IR
819:                           CHR$(y.coord% MOD 256) + _           'lowline
820:                           CHR$(y.coord% \ 256) + _              highline
821:                           CHR$(x.start% MOD 256) + _           'lowX1 (start)
822:                           CHR$(x.start% \ 256) + _             'highX1
823:                           CHR$(x.end%   MOD 256) + _           'lowX2 (end)
824:                           CHR$(x.end%   \ 256) )               'highX2
```

```
825:         CALL PGA.recieve(tmp.string$)
827:         WHILE (tmp.string$<>"")
828:             PUT$ #1, tmp.string$
829:             CALL PGA.error.recieve(tmp.string$)
830:             IF tmp.string$<>"" THEN
831:                 CALL print.error("PGA ERROR: #="+STR$(ASC(tmp.string$)))
832:                 CALL PGA.transmit("CX ")
833:             END IF
834:             DELAY 0.05                          'give PGA time to write data
835:             CALL PGA.recieve(tmp.string$)
836:         WEND
837:     NEXT
838:     CLOSE #1
839:
840:     CALL PGA.transmit("CA DI,0 ")
841:     BEEP
842:
843: END SUB
844:
845:
846:
847:
848: SUB setup.screen(x.offset%, demag, y.offset%, scale)
849:         transmit$ = "WI" + STR$((-%screen.x-x.offset%)* demag) + "," + _
850:                     STR$(( %screen.x-x.offset%)* demag) + "," +
851:                     STR$((-%screen.y-y.offset%)* demag * scale) + "," + _
852:                     STR$(( %screen.y-y.offset%)* demag * scale) + " "
853:         CALL PGA.transmit(transmit$)
854:         CALL PGA.transmit("CLS,0 DI,0 ")
855:         CALL PGA.transmit("TS,"+STR$(8*demag)+" ")
856: END SUB
857:
858:
859:
860:
861: SUB show.help.screen
862:     LOCAL       response$
863:     SHARED      terminate$
864:
865:         SCREEN 0
866:         CLS
867:         COLOR %text.color
868:         PRINT TAB(11); "                                                    "
869:         PRINT TAB(11); "  DEFINITIONS                                       "
870:         PRINT TAB(11); "                                                    "
871:         PRINT TAB(11); "   IMAGE: the fourier descriptor itself.  A descriptor "
872:         PRINT TAB(11); "          is comprised of a fundamental part (harmonic "
873:         PRINT TAB(11); "          = 0) and a nonfundamental part (harmonic is as "
874:         PRINT TAB(11); "          specified by user).                       "
875:         PRINT TAB(11); "   SCREEN: two images combined by either the mkscreen "
876:         PRINT TAB(11); "           program (for FOURIER) or the display routines "
877:         PRINT TAB(11); "           in FOURIER2 or FOURIER3.                 "
878:         PRINT TAB(11); "                                                    "
879:
```

```
880:        PRINT TAB(11); "
882:        PRINT TAB(11); "  FUNCTION KEY DESIGNATIONS                                "
883:        PRINT TAB(11); "
884:        PRINT TAB(11); "  - (minus)          demagnify image                       "
885:        PRINT TAB(11); "  + (plus)           magnify image                         "
886:        PRINT TAB(11); "  INS                increase perspective factor           "
887:        PRINT TAB(11); "  DEL                decrease perspective factor           "
888:        PRINT TAB(11); "  PRTSC              print variable values on printer      "
889:        PRINT TAB(11); "  ESC                terminate program                     "
890:        PRINT TAB(11); "  LEFT ARROW         move image left                       "
891:        PRINT TAB(11); "  RIGHT ARROW        move image right                      "
892:        PRINT TAB(11); "                                                           ";
893:        LOCATE 25, 1
894:        COLOR 15, 1
895:        PRINT STRING$(80, " ");
896:        LOCATE 25, 20
897:        PRINT "PGA graph  v 1.10  By  Christopher Voltz";
898:        DELAY 1.0
899:        LOCATE 25, 1
900:        COLOR 28, 1
901:        PRINT STRING$(80, " ");
902:        LOCATE 25, 21
903:        PRINT "PRESS ANY KEY TO CONTINUE, ESC to skip";
904:        CALL get.key(response$)
905:        IF response$ = terminate$ THEN skip
906:        COLOR %text.color, 0
907:        CLS
908:        PRINT TAB(11); "
909:        PRINT TAB(11); "  FUNCTION KEY DESIGNATIONS (continued)                    "
910:        PRINT TAB(11); "
911:        PRINT TAB(11); "  UP ARROW           move image up                         "
912:        PRINT TAB(11); "  DOWN ARROW         move image down                       "
913:        PRINT TAB(11); "
914:        PRINT TAB(11); "  F1                 display these help pages              "
915:        PRINT TAB(11); "  F2                 decrease digitized image size Y       "
916:        PRINT TAB(11); "  F3                 increase digitized image size Y       "
917:        PRINT TAB(11); "  F4                 select new harmonic frequency         "
918:        PRINT TAB(11); "  F5                 decrease amplitude of harmonic        "
919:        PRINT TAB(11); "  F6                 increase amplitude of harmonic        "
920:        PRINT TAB(11); "  F7                 plot image on HIPLOT DMP-29           "
921:        PRINT TAB(11); "  F8                 toggle fill on/off                    "
922:        PRINT TAB(11); "  F9                 read digitized image                  "
923:        PRINT TAB(11); "  F10                save digitized image in AOI           "
924:        PRINT TAB(11); "  F11 (shift +F1)    change phase shift                    "
925:        PRINT TAB(11); "  F12                decrease digitized image size X       "
926:        PRINT TAB(11); "  F13                increase digitized image size X       "
927:        PRINT TAB(11); "  F14                load in a screen (.SCR file)          "
928:        PRINT TAB(11); "  F15                rotate image (geometrically)          "
929:        PRINT TAB(11); "  F16                make a noise screen (.SCR)            "
930:        PRINT TAB(11); "  F17                enter new amplitude level             "
931:        PRINT TAB(11); "                                                           ";
932:        LOCATE 25, 1
933:        COLOR 28, 1
934:        PRINT STRING$(80, " ");
```

```
935:        LOCATE 25, 21
937:        PRINT "PRESS ANY KEY TO CONTINUE, ESC to skip";
938:        CALL get.key(response$)
939:        IF response$ = terminate$ THEN skip
940:        COLOR %text.color, 0
941:        CLS
942:        PRINT TAB(11); "
943:        PRINT TAB(11); "  FUNCTION KEY DESIGNATIONS (continued)
944:        PRINT TAB(11); "
945:        PRINT TAB(11); "  F19                read two digitized images
946:        PRINT TAB(11); "  F20                execute DOS command
947:        PRINT TAB(11); "
948:        PRINT TAB(11); "  DISPLAY NOTES
949:        PRINT TAB(11); "
950:        PRINT TAB(11); "     At the bottom of the graphic screen is a status
951:        PRINT TAB(11); "    line.  It displays the following information
952:        PRINT TAB(11); "    respectively:
953:        PRINT TAB(11); "    (X,Y)     => the X and Y position of the center of the
954:        PRINT TAB(11); "                 image relative to the center of the screen
955:        PRINT TAB(11); "                 (0,0).  Negative X means to the left and
956:        PRINT TAB(11); "                 negative Y means below.  The units are mm.
957:        PRINT TAB(11); "    DEMAG     => the demagnification factor of the screen.
958:        PRINT TAB(11); "                 Larger numbers make a smaller image.
959:        PRINT TAB(11); "    H         => number of harmonic being displayed.
960:        PRINT TAB(11); "    AMP       => amplitude of harmonic added to the
961:        PRINT TAB(11); "                 fundamental.
962:        PRINT TAB(11); "    PHI.1     => phase angle of harmonic
963:        PRINT TAB(11); "    PHI.2     => phase angle of fundamental
964:        PRINT TAB(11); "    ROT       => angle of geometric rotation of image.
965:        PRINT TAB(11); "
966:        LOCATE 25, 1
967:        COLOR 28, 1
968:        PRINT STRING$(80, " ");
969:        LOCATE 25, 21
970:        PRINT "PRESS ANY KEY TO CONTINUE, ESC to skip";
971:        CALL get.key(response$)
972:        IF response$ = terminate$ THEN skip
973:        COLOR %text.color, 0
974:        CLS
975:        PRINT TAB(11); "
976:        PRINT TAB(11); "  SPECIFIC EXPERIMENT NOTES
977:        PRINT TAB(11); "
978:        PRINT TAB(11); "
979:        PRINT TAB(11); "    DEMAG = 13  => 40 mm (horizontally)
980:        PRINT TAB(11); "    DEMAG =  9  => 60 mm (horizontally)
981:        PRINT TAB(11); "
982:        PRINT TAB(11); "    X = -60     => center of figure 60 mm left of
983:        PRINT TAB(11); "                   screen center
984:        PRINT TAB(11); "    X = +60     => center of figure 60 mm right of
985:        PRINT TAB(11); "                   screen center
986:        PRINT TAB(11); "
987:        PRINT TAB(11); "    BOX.SIZE => size of digitization box in relative
988:        PRINT TAB(11); "                units.  The actual size of the box
989:        PRINT TAB(11); "                changes in direct relation to the
```

```
 990:         PRINT TAB(11); "┌                demagnification factor but its size    ┐"
 992:         PRINT TAB(11); "│                relative to the image is constant at   │"
 993:         PRINT TAB(11); "│                whatever size is chosen.               │"
 994:         PRINT TAB(11); "│    BOX.SIZE.X => 1800                                 │"
 995:         PRINT TAB(11); "│    BOX.SIZE.Y => 2150                                 │"
 996:         PRINT TAB(11); "└                                                       ┘"
 997:         LOCATE 25, 1
 998:         COLOR 28, 1
 999:         PRINT STRING$(80, " ");
1000:         LOCATE 25, 27
1001:         PRINT "PRESS ANY KEY TO CONTINUE";
1002:         CALL get.key(response$)
1003:   skip:
1004:         COLOR %text.color, 0
1005:         CLS
1006: END SUB
```

```
1: TITLE   PGA_transmit_routine
2: PAGE    55, 132
3:
4: COMMENT *
5:  ┌──────────────────────────────────────────────────────────────────────────┐
6:  │                                                                          │
7:  │       FILENAME:                PGAtran.asm                               │
8:  │       PROGRAMMER:              Christopher Voltz - UDRI                  │
9:  │       CREATED:                 -1/8706.11                                │
10: │       LAST MODIFIED:           -1/8706.26                                │
11: │       REQUIREMENTS:            IBM PGA                                   │
12: │       INTERFACE PROTOCALL:     Turbo BASIC                               │
13: │                                                                          │
14: │                                                                          │
15: │       PURPOSE:                                                           │
16: │              This include module allows character strings                │
17: │       to be sent to the Professional Graphics Adapter (PGA)              │
18: │       from a Turbo BASIC program.                                        │
19: │                                                                          │
20: │              The PGA uses a fifo structure to communicate.               │
21: │       Two pointers are maintained, ie. the read and write                │
22: │       pointers.  Both are indices (from 0 to 255) into the               │
23: │       output fifo which is located at an offset of 0000H from            │
24: │       the PGA segment.  When the two pointers are equal, the             │
25: │       queue is full and no writes may take place until the               │
26: │       PGA has caught up.  The queue is checked for room for              │
27: │       a byte to be added by checking to see if the write                 │
28: │       pointer + 1 (modulo 256 because the queue is circular)             │
29: │       is not equal to the read pointer.  If there is room,               │
30: │       the byte is added and the write pointer is updated;                │
31: │       otherwise, a loop is executed to wait until the PGA is             │
32: │       ready.                                                             │
33: │                                                                          │
34: └──────────────────────────────────────────────────────────────────────────┘
35: *
36:
37:
38: ;┌──────────────────────────────────────────────────────────────────────────┐
39: ;│                          LABEL DEFINITIONS                               │
40: ;└──────────────────────────────────────────────────────────────────────────┘
41:
42: PGA_base        EQU     0C600H  ; segment of PGA interface memory
43: read_pointer    EQU     00301H  ; output read pointer
44: write_pointer   EQU     00300H  ; output write pointer
45:
46:
47: ;┌──────────────────────────────────────────────────────────────────────────┐
48: ;│                        STRUCTURE DEFINITIONS                             │
49: ;└──────────────────────────────────────────────────────────────────────────┘
50:
51: stack_struc     STRUC
52:         old_BP          DW      ?       ; old base pointer
53:         return_address  DD      ?       ; far return address
54:         string          DD      ?       ; 32-bit pointer to string
```

```
stack_struc     ENDS


;
;                       PGA TRANSMIT PROCEDURE
;

program SEGMENT

        ASSUME CS:program

        PUSH    BP                      ;setup stack addressing
        MOV     BP, SP

        PUSH    DS                      ;save segment registers
        PUSH    ES

        LES     SI, [BP].string         ;get pointer to string descriptor
        MOV     CX, ES:[SI]             ;get string length
        AND     CH, 01111111B           ;clear high bit
        MOV     SI, ES:[SI+2]           ;get offset of first character
        MOV     ES, DS:[0]              ;get segment of strings
        MOV     BX, PGA_base            ;get PGA segment
        MOV     DS, BX
        MOV     DI, write_pointer       ;get offset of output write pointer
        SUB     BH, BH                  ;zero BH, BX will point to destination

 jmp_1:
        MOV     BL, [DI]                ;get pointer to end of queue
        INC     BL                      ;check if room for one more byte
        CMP     BL, DS:[read_pointer]   ;by checking if (WP + 1) mod 256 is
        JE      SHORT jmp_1             ;not equal to RP,  wait if no room
        MOV     AL, ES:[SI]             ;otherwise, move byte from string
        DEC     BL
        MOV     [BX], AL                ;to PGA queue
        INC     BYTE PTR [DI]           ;update write pointer
        INC     SI                      ;point to next character
        LOOP    jmp_1                   ;repeat until CX=0

        POP     ES                      ;restore segment registers
        POP     DS

        POP     BP                      ;restore stack

program ENDS

        END
```

```
1: TITLE   PGA_recieve_routine
2: PAGE    55, 132
3:
4: COMMENT *
5: ┌──────────────────────────────────────────────────────────────────────┐
6: │                                                                      │
7: │       FILENAME:                PGArecv.asm                           │
8: │       PROGRAMMER:              Christopher Voltz - UDRI              │
9: │       CREATED:                 -1/8706.18                            │
10:│       LAST MODIFIED:           -1/8711.07                            │
11:│       REQUIREMENTS:            IBM PGA                               │
12:│       INTERFACE PROTOCALL:     Turbo BASIC                           │
13:│                                                                      │
14:│                                                                      │
15:│       PURPOSE:                                                       │
16:│              This include module allows character strings            │
17:│       to be read from the Professional Graphics Adapter (PGA)        │
18:│       from a Turbo BASIC program.                                    │
19:│                                                                      │
20:│              The PGA uses a fifo structure to communicate.           │
21:│       Two pointers are maintained, ie. the read and write            │
22:│       pointers.  Both are indices (from 0 to 255) into the           │
23:│       input fifo which is located at an offset of 0100H from         │
24:│       the PGA segment.  When the two pointers are equal, the         │
25:│       queue is empty and no reads may take place until the           │
26:│       PGA has sent data.  If the queue is not empty,                 │
27:│       characters are read one at a time and placed into the          │
28:│       string until there are no more characters to be read or        │
29:│       the string is full.                                            │
30:│                                                                      │
31:│              This routine requires that a string be sent to          │
32:│       it which has been filled with some character (this             │
33:│       routine is not allowed to alter the string length, only        │
34:│       its contents; so if a string of length=256 is sent, a          │
35:│       maximum of 256 bytes may be read.).  Also, an integer          │
36:│       variable must be passed to determine the number of bytes       │
37:│       actually returned.                                             │
38:│                                                                      │
39:│                                                                      │
40:└──────────────────────────────────────────────────────────────────────┘
41: *
42:
43:
44: ;┌──────────────────────────────────────────────────────────────────────┐
45: ;│                          LABEL DEFINITIONS                           │
46: ;└──────────────────────────────────────────────────────────────────────┘
47:
48: PGA_base        EQU     0C600H  ; segment of PGA interface memory
49: queue_base      EQU     00100H  ; input queue base
50: read_pointer    EQU     00303H  ; input read pointer
51: write_pointer   EQU     00302H  ; input write pointer
52:
53:
54: ;┌──────────────────────────────────────────────────────────────────────┐
```

```
55: ;┌──────────────────────────────────────────────────────────────┐
57: ;└──────────────────────────────────────────────────────────────┘
58:
59: stack_struc     STRUC
60:         old_BP          DW      ?       ; old base pointer
61:         return_address  DD      ?       ; far return address
62:         string          DD      ?       ; 32-bit pointer to string       (I/O)
63:         slength         DD      ?       ; 32-bit pointer to string length (O)
64: stack_struc     ENDS
65:
66:
67:
68: ;┌──────────────────────────────────────────────────────────────┐
69: ;│                   PGA RECIEVE PROCEDURE                      │
70: ;└──────────────────────────────────────────────────────────────┘
71:
72: program SEGMENT
73:
74:         ASSUME CS:program
75:
76:         PUSH    BP                      ;setup stack addressing
77:         MOV     BP, SP
78:
79:         PUSH    DS                      ;save segment registers
80:         PUSH    ES
81:
82:         SUB     DX, DX                  ;set "number of bytes read" to zero
83:         LES     DI, [BP].string         ;get pointer to string descriptor
84:         MOV     CX, ES:[DI]             ;get string length
85:         AND     CH, 01111111B           ;clear high bit
86:         JCXZ    SHORT jmp_2             ;if null string, return
87:         MOV     DI, ES:[DI+2]           ;get offset of first character
88:         MOV     ES, DS:[0]              ;get segment of strings
89:         MOV     BX, PGA_base            ;get PGA segment
90:         MOV     DS, BX
91:         MOV     SI, read_pointer        ;get offset of input read pointer
92:         SUB     BH, BH                  ;zero BH, BX will point to source
93:
94:  jmp_1:
95:         MOV     BL, [SI]                ;get pointer to start of queue
96:         CMP     BL, DS:[write_pointer]  ;check if there is a byte to read
97:         JE      SHORT jmp_2             ;if equal, no bytes to read, we are done
98:         MOV     AL, [BX+queue_base]     ;otherwise, move byte from queue
99:         MOV     ES:[DI], AL             ;to string
100:        INC     DI                      ;point to next character
101:        INC     BYTE PTR [SI]           ;update read pointer
102:        INC     DX                      ;update "number of bytes read"
103:        LOOP    jmp_1                   ;repeat until string is full
104:
105: jmp_2:
106:        LDS     SI, [BP].slength        ;get pointer to string length
107:        MOV     [SI], DX                ;return "number of bytes read" in length
108:        POP     ES                      ;restore segment registers
109:        POP     DS
```

```
110:
112:            POP     BP                      ;restore stack
113:
114: program ENDS
115:
116:            END
```

```
 1: TITLE   PGA_error_recieve_routine
 2: PAGE    55, 132
 3:
 4: COMMENT *
 5: ┌──────────────────────────────────────────────────────────────────────┐
 6: │                                                                      │
 7: │       FILENAME:               PGAerrcv.asm                           │
 8: │       PROGRAMMER:             Christopher Voltz - UDRI               │
 9: │       CREATED:                -1/8706.18                             │
10: │       LAST MODIFIED:          -1/8711.07                             │
11: │       REQUIREMENTS:           IBM PGA                                │
12: │       INTERFACE PROTOCALL:    Turbo BASIC                            │
13: │                                                                      │
14: │                                                                      │
15: │       PURPOSE:                                                       │
16: │             This include module allows error messages to be          │
17: │       read from the Professional Graphics Adapter (PGA) from         │
18: │       a Turbo BASIC program.                                         │
19: │                                                                      │
20: │             The PGA uses a fifo structure to communicate.            │
21: │       Two pointers are maintained, ie. the read and write            │
22: │       pointers.  Both are indices (from 0 to 255) into the           │
23: │       error fifo which is located at an offset of 0200H from         │
24: │       the PGA segment.  When the two pointers are equal, the         │
25: │       queue is empty and no reads may take place until the           │
26: │       PGA has sent data.  If the queue is not empty,                 │
27: │       characters are read one at a time and placed into the          │
28: │       string until there are no more characters to be read or        │
29: │       the string is full.                                            │
30: │                                                                      │
31: │             This routine requires that a string be sent to           │
32: │       it which has been filled with some character (this             │
33: │       routine is not allowed to alter the string length, only        │
34: │       its contents; so if a string of length=256 is sent, a          │
35: │       maximum of 256 bytes may be read.).  Also, an integer          │
36: │       variable must be passed to determine the number of bytes       │
37: │       actually returned.                                             │
38: │                                                                      │
39: └──────────────────────────────────────────────────────────────────────┘
40: *
41:
42:
43: ;┌──────────────────────────────────────────────────────────────────────┐
44: ;│                          LABEL DEFINITIONS                           │
45: ;└──────────────────────────────────────────────────────────────────────┘
46:
47: PGA_base        EQU     0C600H  ; segment of PGA interface memory
48: queue_base      EQU     00200H  ; error queue base
49: read_pointer    EQU     00305H  ; error read pointer
50: write_pointer   EQU     00304H  ; error write pointer
51:
52:
53: ;┌──────────────────────────────────────────────────────────────────────┐
54: ;│                        STRUCTURE DEFINITIONS                         │
```

```
55: ;┌──────────────────────────────────────────────────────────────┐
57:
58: stack_struc     STRUC
59:         old_BP          DW      ?       ; old base pointer
60:         return_address  DD      ?       ; far return address
61:         string          DD      ?       ; 32-bit pointer to string        (I/O)
62:         slength         DD      ?       ; 32-bit pointer to string length (O)
63: stack_struc     ENDS
64:
65:
66:
67: ;┌──────────────────────────────────────────────────────────────┐
68: ;│                  PGA ERROR RECIEVE PROCEDURE                 │
69: ;└──────────────────────────────────────────────────────────────┘
70:
71: program SEGMENT
72:
73:         ASSUME CS:program
74:
75:         PUSH    BP                      ;setup stack addressing
76:         MOV     BP, SP
77:
78:         PUSH    DS                      ;save segment registers
79:         PUSH    ES
80:
81:         SUB     DX, DX                  ;set "number of bytes read" to zero
82:         LES     DI, [BP].string         ;get pointer to string descriptor
83:         MOV     CX, ES:[DI]             ;get string length
84:         AND     CH, 01111111B           ;clear high bit
85:         JCXZ    SHORT jmp_2             ;if null string, return
86:         MOV     DI, ES:[DI+2]           ;get offset of first character
87:         MOV     ES, DS:[0]              ;get segment of strings
88:         MOV     BX, PGA_base            ;get PGA segment
89:         MOV     DS, BX
90:         MOV     SI, read_pointer        ;get offset of input read pointer
91:         SUB     BH, BH                  ;zero BH, BX will point to source
92:
93:  jmp_1:
94:         MOV     BL, [SI]                ;get pointer to start of queue
95:         CMP     BL, DS:[write_pointer]  ;check if there is a byte to read
96:         JE      SHORT jmp_2             ;if equal, no bytes to read
97:         MOV     AL, [BX+queue_base]     ;otherwise, move byte from queue
98:         MOV     ES:[DI], AL             ;to string
99:         INC     DI                      ;point to next character
100:        INC     BYTE PTR [SI]           ;update read pointer
101:        INC     DX                      ;update "number of bytes read"
102:        LOOP    jmp_1                   ;repeat until string is full
103:
104: jmp_2:
105:        LDS     SI, [BP].slength        ;get pointer to string length
106:        MOV     [SI], DX                ;return "number of bytes read" in length
107:        POP     ES                      ;restore segment registers
108:        POP     DS
109:
```

```
110:          POP     BP                      ;restore stack
112:
113: program ENDS
114:
115:          END
```

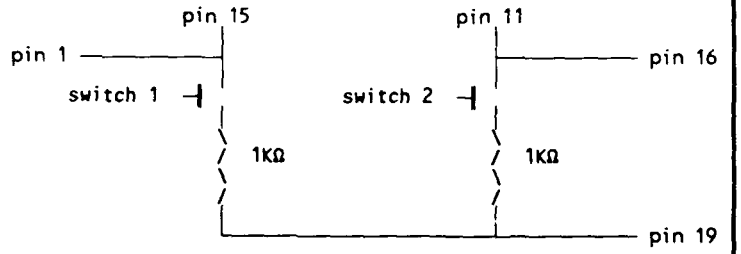
```
 1: 10 CLS
 2: 20 PI=3.1415927#
 3: 30 '
 4: 40 DIM DIX(300),DIY(300),VIY(300),VIX(300),E(61),A(300),T(300),DT(300)
 5: 50 DIM DX(300),DY(300),XP(300),YP(300),I(300),IY(300),AN(61),AN2(300)
 6: 60 DIM BN(61),BN2(300),CN(61),DN(61),CN2(300),DN2(300),AN1(300),AN3(300)
 7: 70 DIM EP(300),LP(300),EX2(61),EY2(61),EVR(61),AE(61),AEY(61),ABE(61)
 8: 80 DIM AON(300),CON(300),X(201),Y(201),X2(61),Y2(61),VR(61),DXP(300)
 9: 90 DIM DTP(300),DYP(300),BN1(300),BN3(300),I2(300),IY2(300),T2(300)
10: 200 CLS
11: 210 LOCATE 1,20 : PRINT "Data Input - Contour Parameters"
12: 220 LOCATE 3,1
13: 230 INPUT "Enter the number of links in the chain code...: ",K
14: 240 PRINT ""
15: 250 PRINT "Starting with the orgin, enter the number, then"
16: 260 PRINT "the value of the links (Links Range From 0 to 7)."
17: 270 T=0 : NUM1=0
18: 280 PRINT ""
19: 290 INPUT "Enter the number of links and the value of the links...: ',N,VALUE
20: 300 NUM=T+N
21: 310 FOR I=NUM1+1 TO NUM
22: 320 A(I)=VALUE:PRINT "link" I"=" A(I)
23: 330 IF (A(I) >7) THEN PRINT "exceeded link value": GOTO 320
24: 340 IF A(I)=0 OR A(I)=2 OR A(I)=4 OR A(I)=6 THEN DT(I)=1 ELSE DT(I)=SQR(2)
25: 350 DX(I)=(SGN(6-A(I)))*(SGN(2-A(I)))
26: 360 DY(I)=(SGN(4-A(I)))*(SGN(A(I)))
27: 370 IF (I=NUM) AND (K-NUM=0) THEN GOTO 400 ELSE GOTO 380
28: 380 IF (I=NUM)AND (K-NUM>0) THEN NUM1=NUM:T=NUM:GOTO 290
29: 390 NEXT I
30: 400 CLS
31: 410 LOCATE 12,25: PRINT "Analysis Phase - Please Wait"
32: 420 T(1)=DT(1)
33: 430 XP(1)=DX(1)
34: 440 YP(1)=DY(1)
35: 450 FOR I=2 TO K
36: 460 T(I)=DT(I)+T(I-1)
37: 470 XP(I)=DX(I)+XP(I-1)
38: 480 YP(I)=DY(I)+YP(I-1)
39: 490 DXP(I)=XP(I)-XP(I-1)
40: 500 DYP(I)=YP(I)-YP(I-1)
41: 510 NEXT I
42: 520 FOR I=1 TO K
43: 530 DIX(I)=DX(I)/DT(I):DIY(I)=DY(I)/DT(I)
44: 540 NEXT I
45: 550 VIX(1)=0:VIY(1)=0
46: 560 FOR I=2 TO K
47: 570 VIY(I)=(DIY(I)-DIY(I-1)) + VIY(I-1)
48: 580 VIX(I)=(DIX(I)-DIX(I-1)) + VIX(I-1)
49: 590 NEXT I
50: 600 VXT = ABS(VIX(K) + ABS(DIX(K)-DIX(1)))
51: 610 VYT = ABS(VIY(K) + ABS(DIY(K)+DIY(1)))
52: 620 IF VXT > VYT THEN M = VXT
53: 630 IF VYT > VXT THEN M = VYT
54: 640 IF VYT = VXT THEN M = VYT

FILE=mkgrdata.bas     Wed Jun 14 16:23:01 1989     PAGE=1
```

```
 55: 650 FOR N=1 TO 60
 57: 660 E(N)=(T(K)/(2*(PI^2)*N))*M
 58: 670 NEXT N
 59: 680 HA=61
 60: 690 FOR I=1 TO K
 61: 700 DTP(I)= SQR((DX(I)^2)+(DY(I)^2))
 62: 710 NEXT I
 63: 720 AN2(1)=0:BN2(1)=0:CN2(1)=0:DN2(1)=0 :T(1)=0
 64: 730 FOR N=1 TO HA-1
 65: 740 FOR I=2 TO K
 66: 750 AN1(I)=COS((2*PI*N*T(I))/T(K))
 67: 760 BN1(I)=SIN((2*PI*N*T(I))/T(K))
 68: 770 AN3(I)=COS((2*PI*N*T(I-1))/T(K))
 69: 780 BN3(I)=SIN((2*PI*N*T(I-1))/T(K))
 70: 790 AN2(I)=((DXP(I)/DTP(I))*(AN1(I)-AN3(I)))+AN2(I-1)
 71: 800 BN2(I)=((DXP(I)/DTP(I))*(BN1(I)-BN3(I)))+BN2(I-1)
 72: 810 CN2(I)=((DYP(I)/DTP(I))*(AN1(I)-AN3(I)))+CN2(I-1)
 73: 820 DN2(I)=((DYP(I)/DTP(I))*(BN1(I)-BN3(I)))+ DN2(I-1)
 74: 830 NEXT I
 75: 840 F = T(K)/(2*(N^2)*(PI^2))
 76: 850 AN(N)=F * AN2(K)
 77: 860 BN(N)=F * BN2(K)
 78: 870 CN(N)=F * CN2(K)
 79: 880 DN(N)=F * DN2(K)
 80: 890 NEXT N
 81: 900 S=1 : EX2(0) = 0 : EY2(0) = 0
 82: 910 FOR N=1 TO 60
 83: 920 EVR(N) = (2*PI*N*S)/T(K)
 84: 930 EX2(N) = AN(N)*COS(EVR(N)) + BN(N)*SIN(EVR(N)) + EX2(N-1)
 85: 940 EY2(N) = CN(N)*COS(EVR(N)) + DN(N)*SIN(EVR(N)) + EY2(N-1)
 86: 950 NEXT N
 87: 960 FOR N=1 TO 59
 88: 970 AE(N) = ABS(EX2(60)-EX2(N)) : AEY(N) = ABS(EY2(60)-EY2(N))
 89: 980 IF (AE(N)>AEY(N)) THEN ABE(N) = AE(N) ELSE ABE(N) = AEY(N)
 90: 990 NEXT N
 91: 1000 CLS : BEEP
 92: 1010 INPUT "Would you like to see the error level of each harmonic? (Type y or n) ";ANS$
 93: 1020 IF ANS$ = "y" OR ANS$ = "Y" THEN GOTO 1030 ELSE 1210
 94: 1030 CLS
 95: 1040 FOR N=1 TO 22
 96: 1050 PRINT USING "## Actual error = ##.#####    Predicted error = ##.######";N,ABE(N),E(N)
 97: 1060 NEXT N
 98: 1070 PRINT ""
 99: 1080 INPUT "Would you like to see more? (Type y or n)..: ",ANS$
100: 1090 IF ANS$ = "y" OR ANS$ = "Y" THEN GOTO 1100 ELSE GOTO 1210
101: 1100 CLS
102: 1110 FOR N=23 TO 44
103: 1120 PRINT USING "## Actual error = ##.#####    Predicted error = ##.######";N,ABE(N),E(N)
104: 1130 NEXT N
105: 1140 PRINT ""
106: 1150 INPUT "Would you like to see more? (Type y or n)..: ",ANS$
107: 1160 IF ANS$ = "y" OR ANS$ = "Y" THEN GOTO 1170 ELSE GOTO 1210
108: 1170 CLS
109: 1180 FOR N=45 TO 60
```

```
110: 1190 PRINT USING "## Actual error = ##.#####    Predicted error = ##.######";N,ABE(N),E(N)
112: 1200 NEXT N
113: 1210 EP(1)=0 : LP(1)=0
114: 1220 PRINT ""
115: 1230 INPUT "Would you like to see the magnitude of the all the harmonics? (Type y or n) ";ANS$
116: 1240 IF ANS$ = "y" OR ANS$ = "Y" THEN GOTO 1250 ELSE 1470
117: 1250 CLS
118: 1260 FOR N=1 TO 22
119: 1270 MAG = LOG(SQR(AN(N)^2 +BN(N)^2)) : MAG2=LOG(SQR(CN(N)^2+DN(N)^2))
120: 1280 PRINT USING "##  log(SQR(A^2 +B^2)) = ##.######   log(SQR(C^2+D^2)) = ##.######";N,MAG,MAG2
121: 1290 NEXT N
122: 1300 PRINT ""
123: 1310 INPUT "Would you like to see more? (Type y or n)...: ",ANS$
124: 1320 IF ANS$ = "y" OR ANS$ = "Y" THEN GOTO 1330 ELSE 1470
125: 1330 CLS
126: 1340 FOR N=23 TO 44
127: 1350 MAG = LOG(SQR(AN(N)^2 +BN(N)^2)) : MAG2=LOG(SQR(CN(N)^2+DN(N)^2))
128: 1360 PRINT USING "##  log(SQR(A^2 +B^2)) = ##.######   log(sqr(C^2+D^2)) = ##.######";N,MAG,MAG2
129: 1370 NEXT N
130: 1380 PRINT ""
131: 1390 INPUT "Would you like to see more? (Type y or n)...: ",ANS$
132: 1400 IF ANS$ = "y" OR ANS$ = "Y" THEN GOTO 1410 ELSE 1470
133: 1410 CLS
134: 1420 FOR N=45 TO 60
135: 1430 MAG = LOG(SQR(AN(N)^2 +BN(N)^2)) : MAG2=LOG(SQR(CN(N)^2+DN(N)^2))
136: 1440 PRINT USING "##  log(SQR(A^2 +B^2)) = ##.######  log(SQR(C^2+D^2)) = ##.######";N,MAG,MAG2
137: 1450 NEXT N
138: 1460 '
139: 1470 PRINT ""
140:
141:      INPUT "What filename should I save the data under? (.DAT) ", filename$
142:      OPEN filename$+".DAT" FOR OUTPUT AS #1
143:      PRINT #1, k
144:      FOR index = 1 to K 'number of links in chain code
145:          PRINT #1, T(index)
146:      NEXT
147:      FOR index = 1 to 60
148:          PRINT #1, AN(index), BN(index), CN(index), DN(index)
149:      NEXT
150:      CLOSE #1
151:
```

```
1: '
2: '
3: '     FILENAME:               response.inc
4: '     PROGRAMMER:             Christopher Voltz - UDRI
5: '     CREATED:                -1/8712.08
6: '     LAST MODIFIED:          -1/8801.11
7: '     TARGET:                 IBM PC w/ I/O port
8: '     INTERFACE PROTOCALL:    TURBO BASIC v. 1.10
9: '     REQUIRED FILES:         none
10: '
11: '
12: '
13: '          This include module allows the user to read the two
14: '     buttons connected through the response box to the parallel port
15: '     designated by the main program.  The included routines are:
16: '     1) GET.RESPONSE =>  this routine reads data from the port, ANDs
17: '        it against the given bitmask, goes into a loop where it
18: '        continously reads data from the port and ANDs it to the
19: '        bitmask until the new data is different from the old data
20: '        (ie. a change in states is detected), XORs the new data with
21: '        the old data to set the bits which have changed and returns
22: '        that byte of data.  It is called with an integer parameter in
23: '        which to return the resulting byte of data.
24: '        eg.: CALL get.response(response%)
25: '        NOTE: if any key is pressed on the keyboard, this routine
26: '        will terminate immediately without removing the keystroke
27: '        from the keyboard buffer.
28: '     2) TEST.RESPONSE.BOX => this routine reads a response from the
29: '        response box, compares it to the two legal values the result
30: '        may have (same or different) and prints a message indicating
31: '        which button was pressed (same or different) or an error
32: '        message if the bit pattern was unrecognized.
33: '
34: '          The main program must initialize the following
35: '     variables as required:
36: '     %BIT.MASK     => the bit pattern to AND the input data with; used
37: '       to clear extra bits so only relevant bits are set or reset,
38: '       eg.: %bit.mask = &B10001000    'for box which responds such
39: '                                      'that bits 7 and 3 are set
40: '     %DIFFERENT.BUTTON => the bit pattern which represents the
41: '       different button being pressed,
42: '       eg.: %different.button = &B10000000 'if bit 7 represents
43: '                                           'different
44: '     %SAME.BUTTON => the bit pattern which represents the same button
45: '       being pressed,
46: '       eg.: %same.button = &B00001000 'if bit 3 represents same
47: '     %SWITCH.PORT => the address of the port to read the data from,
48: '       eg.: %switch.port = &H379      'for LPT1:
49: '
50: '          Note that due to the method used to read the data, any
51: '     type of switch and/or port combination may be used if the pins
52: '     representing the status of the buttons are constantly driven.
53: '     That is, a serial or a parallel port may be used with equal
54: '     ease if the correct data port addresses are given.  Also, if
```

```
'       the button is normally open, normally closed, or momentary it
'       will interface with these routines as they detect the change in
'       states which must result from a button closure; however, if the
'       button is momentary, the routines could respond with two results
'       if the user open and closed (or vice versa) the switches slow
'       enough that these routines were called again.
'       eg.: two momentary switches shall be connected to a parallel
'            port, a sample circut might be:
```




```
'       recall:  pin 1  => -STROBE (provides +5V (high) for switch 1)
'                pin 11 => BUSY    (input for switch 1 => bit 7 of
'                                    status word)
'                pin 15 => -ERROR  (input for switch => bit 3 of
'                                    status word)
'                pin 16 => -INIT   (provides +5V (high) for switch 2)
'                pin 19 => GROUND  (provides ground (low) for switches)
'
'              If we wished to connect this to LPT1: and we wish
'       switch 1 to represent the SAME button and switch 2 to represent
'       the DIFFERENT button, then we would define the system parameters
'       as shown in the previous examples for setting them.
'
'
'
'
'
'
'       GET.RESPONSE
'
'          This module reads the change in states from the response box
'       connected to the %switch.port.  It returns with the bits set to
'       one which correspond to bits which have been changed.
'
'
SUB get.response(response%)
'
    LOCAL       state.1%                'initial state of switches
    LOCAL       state.2%                'final state of switches
```

```
110:
112:     state.1% = INP(%switch.port)                'get initial state
113:     state.1% = state.1% AND %bit.mask           'clear extra bits
114:     DO
115:         state.2% = INP(%switch.port)                    'get next state
116:         state.2% = state.2% AND %bit.mask               'clear extra bits
117:     LOOP UNTIL (state.1% <> state.2%) OR (INKEY$<>"")   'wait for a change or a keypress
118:     response% = state.1% XOR state.2%                   'return change
119:
120: END SUB
121:
122:
123:
124:
125: '
126: '   TEST.RESPONSE.BOX
127: '
128: '
129: '               This module tests the response box to make sure it is
130: '       functioning correctly.
131: '
132: '
133:
134: SUB test.response.box
135:
136:         '**** LOCAL VARIABLE DECLARATIONS
137:     LOCAL       response%
138:
139:
140:         ' *** setup screen
141:     CLS
142:     PRINT "TESTING SWITCHBOX: (connected to LPT1:)"
143:     PRINT
144:     PRINT "Press any keyboard key to end"
145:     PRINT "Press any switchbox key to see response"
146:     PRINT
147:     PRINT "BEGIN:"
148:
149:     COLOR %option.color
150:     WHILE (INKEY$="")
151:         CALL get.response(response%)
152:         IF (response%=%same.button) THEN
153:             PRINT "     SAME"
154:         ELSEIF (response%=%different.button) THEN
155:             PRINT "     DIFFERENT"
156:         ELSE
157:             COLOR %error.color
158:             PRINT "     ERROR: (switch pattern=";BIN$(response%);"B)"
159:             COLOR %option.color
160:         END IF
161:     WEND
162:
163: END SUB
```

```
1:  '
2:  '
3:  '     FILENAME:              toolbox.inc
4:  '     PROGRAMMER:            Christopher Voltz - UDRI
5:  '     CREATED:               -1/8801.05
6:  '     LAST MODIFIED:         -1/8802.12
7:  '     INTERFACE PROTOCOL:    Turbo BASIC v 1.1
8:  '
9:  '
10: '
11: '
12: '                    MODULE PURPOSE
13: '
14: '
15: '
16: '
17: '        This module is a collection of commonly used routines.  To
18: '     include these routines enter the following line in the main
19: '     program's code:
20: '         $INCLUDE "toolbox.inc"
21: '     By including this module you have access to the following
22: '     routines:
23: '     1) CONFIRM ==> this routine prints an error message, waits for
24: '        the user to press a key, erases the message, and returns the
25: '        key pressed.
26: '        eg.: CALL confirm(response$)
27: '     2) GET.ERROR.MESSAGE$ ==> this function returns the error
28: '        message associated with the most recent error.
29: '        eg.: error.message$ = FN get.error.message$
30: '     3) GET.KEY ==> this routine reads a keypress from the keyboard
31: '        and returns it.
32: '        eg.: CALL get.key(response$)
33: '     4) PARSE ==> this routine takes a string which represents
34: '        command line options (delimited by a stroke "/") and returns
35: '        the first option and the input string without the first
36: '        option.  Use this routine in sucessive calls to parse the
37: '        entire command string.  White space is insignifigant.
38: '        eg.:  if in.string$ = "/c /l"
39: '                 CALL parse(in.string$, out.string$)
40: '                 results in in.string$ = "/l", out.string$ = "c"
41: '     5) PRINT.ERROR ==> this routine prints the given message, beeps,
42: '        waits for the user to press the ESCape key, erases the
43: '        message, and returns.
44: '        eg.: CALL print.error("You idiot!  You hit an invalid key.")
45: '     6) PRINT.OPTION ==> this routine recieves a string which
46: '        contains two strings separated by a pipe "|".  The second
47: '        string is the text to be printed and the first string is the
48: '        text, found within the text to be printed, which is to be
49: '        highlighted.  This routine is case sensitive.
50: '        eg.: CALL print.option("D|Collect Data")
51: '
52: '         The following indentifiers must be defined before these
53: '     routines are called:
54: '     %entry.indent   eg.: %entry.indent = 5  'tab messages by 5
```

```
55: '|      %error.color    eg.: %error.color  = 4  'errors are in red
57: '|      %menu.color     eg.: %menu.color   = 7  'normal text in white
58: '|      %menu.indent    eg.: %menu.indent  = 1  'tab menus by 1
59: '|      %option.color   eg.: %option.color = 3  'highlight in cyan
60: '|
61: '+-------------------------------------------------------------------
62:
63:
64:
65:
66: '+-------------------------------------------------------------------
67: '|      CONFIRM
68: '|
69: '|
70: '|         This module is used to confirm that the user wishes to do
71: '|      something.  It prints the confirm message in the error color,
72: '|      beeps, and waits for the user to press a key.  It then erases
73: '|      the message, returns the cursor to its original position and
74: '|      returns to the calling module.
75: '|
76: '+-------------------------------------------------------------------
77:
78: SUB confirm(out.string$)
79:
80:         ' **** VARIABLE DECLARATIONS
81:     LOCAL col%          'column cursor was on when called
82:     LOCAL row%          'row cursor was on when called
83:     LOCAL temp$         'temporary string variable
84:
85:
86:         ' *** save cursor position
87:     row% = CSRLIN
88:     col% = POS
89:
90:         ' *** print confirm message and alert user
91:     PRINT
92:     PRINT
93:     PRINT TAB(%entry.indent);
94:     COLOR %error.color
95:     BEEP
96:     PRINT "CONFIRM (Y/N): ";
97:     COLOR %menu.color
98:
99:         ' *** wait for user to press a key
100:    DO                                  'clear keyboard buffer
101:        response$ = INKEY$
102:    LOOP UNTIL response$=""
103:
104:    CALL get.key(out.string$)           'get keystroke
105:
106:        ' *** clear confirm message
107:    temp$ = STRING$(80*(CSRLIN-row%+1), " ")
108:    LOCATE row%, col%
109:    PRINT temp$;
```

```
110:     LOCATE row%, col%
112:
113: END SUB
114:
115:
116:
117:
118: '
119: '  GET.ERROR.MESSAGE$
120: '
121: '
122: '                This function returns the error message associated with
123: '       the most recent error.
124: '
125: '
126:
127: DEF FN get.error.message$
128:
129:     SELECT CASE ERR
130:         CASE 2
131:             FN get.error.message$ = "Syntax error"
132:         CASE 3
133:             FN get.error.message$ = "RETURN without GOSUB"
134:         CASE 4
135:             FN get.error.message$ = "Out of data"
136:         CASE 5
137:             FN get.error.message$ = "Illegal function call"
138:         CASE 6
139:             FN get.error.message$ = "Overflow"
140:         CASE 7
141:             FN get.error.message$ = "Out of memory"
142:         CASE 9
143:             FN get.error.message$ = "Subscript out of range"
144:         CASE 10
145:             FN get.error.message$ = "Duplicate definition"
146:         CASE 11
147:             FN get.error.message$ = "Division by zero"
148:         CASE 13
149:             FN get.error.message$ = "Type mismatch"
150:         CASE 14
151:             FN get.error.message$ = "Out of string space"
152:         CASE 15
153:             FN get.error.message$ = "String too long"
154:         CASE 19
155:             FN get.error.message$ = "No RESUME"
156:         CASE 20
157:             FN get.error.message$ = "RESUME without error"
158:         CASE 24
159:             FN get.error.message$ = "Device time-out"
160:         CASE 25
161:             FN get.error.message$ = "Device fault"
162:         CASE 27
163:             FN get.error.message$ = "Out of paper"
164:         CASE 50
```

```
165:            FN get.error.message$ = "Field overflow"
167:        CASE 51
168:            FN get.error.message$ = "Internal error"
169:        CASE 52
170:            FN get.error.message$ = "Bad file number"
171:        CASE 53
172:            FN get.error.message$ = "File not found"
173:        CASE 54
174:            FN get.error.message$ = "Bad file mode"
175:        CASE 55
176:            FN get.error.message$ = "File already open"
177:        CASE 57
178:            FN get.error.message$ = "Device I/O error"
179:        CASE 58
180:            FN get.error.message$ = "File already exists"
181:        CASE 61
182:            FN get.error.message$ = "Disk full"
183:        CASE 62
184:            FN get.error.message$ = "Input past end"
185:        CASE 63
186:            FN get.error.message$ = "Bad record number"
187:        CASE 64
188:            FN get.error.message$ = "Bad file name"
189:        CASE 67
190:            FN get.error.message$ = "Too many files"
191:        CASE 68
192:            FN get.error.message$ = "Device unavailable"
193:        CASE 69
194:            FN get.error.message$ = "Communications buffer overflow"
195:        CASE 70
196:            FN get.error.message$ = "Permission denied"
197:        CASE 71
198:            FN get.error.message$ = "Disk not ready"
199:        CASE 72
200:            FN get.error.message$ = "Disk media error"
201:        CASE 74
202:            FN get.error.message$ = "Rename across disks"
203:        CASE 75
204:            FN get.error.message$ = "Path/File access error"
205:        CASE 76
206:            FN get.error.message$ = "Path not found"
207:        CASE 202
208:            FN get.error.message$ = "Out of string temp space"
209:        CASE 203
210:            FN get.error.message$ = "Mismatched common variables"
211:        CASE 204
212:            FN get.error.message$ = "Mismatch program options"
213:        CASE 205
214:            FN get.error.message$ = "Mismatched program revisions"
215:        CASE 206
216:            FN get.error.message$ = "Invalid program file"
217:        CASE 242
218:            FN get.error.message$ = "String memory corrupt"
219:        CASE 243
```

```
220:             FN get.error.message$ = "CHAIN/RUN from .EXE file only"
222:         CASE ELSE
223:             FN get.error.message$ = ""
224:     END SELECT
225:
226: END DEF
227:
228:
229:
230:
231: '
232: '       GET.KEY
233: '
234: '
235: '           This module waits for a key to be pressed, reads it, and
236: '       returns it to the calling program.  The keypress is NOT echoed.
237: '
238: '
239:
240: SUB get.key(response$)
241:
242:     WHILE NOT INSTAT    'wait until a key is pressed
243:     WEND
244:     response$ = INKEY$
245:
246: END SUB
247:
248:
249:
250: '
251: '       PARSE
252: '
253: '
254: '           This module takes a string which represents options
255: '       preceeded by "/".  It returns the first option in action$.  The
256: '       rest of the string is returned in the variable it was sent in,
257: '       comm$.  NOTE: the initial "/" is not returned.
258: '
259: '
260:
261: SUB parse(comm$, action$)
262:
263:         '**** VARIABLE DECLARATIONS
264:     LOCAL       index%
265:
266:     index% = INSTR(1, comm$, "/")                   'find first / (skip leading spaces)
267:     IF (index%=0) OR (comm$="") THEN                'if none
268:         action$ = ""                                '  no action
269:         comm$   = ""                                '  clear command$
270:     ELSE
271:         action$ = MID$(comm$, index%+1)             'get right part of $
272:         index% = INSTR(1, action$, " ")             'find end of command
273:         IF (index%=0) THEN                          'if not " "
274:             index% = INSTR(2, action$, "/")         '  look for /
```

```
            IF (index%=0) THEN                          'if no /
                comm$ = ""                              '  last command
            ELSE
                action$ = LEFT$(action$, index%-1)      'get first command
                comm$ = MID$(comm$, index%)             'save rest of commands
            END IF
        ELSE
            action$ = LEFT$(action$, index%-1)          'get first command
            comm$ = MID$(comm$, index%)                 'save rest of commands
        END IF
    END IF

END SUB



'
'       PRINT.ERROR
'
'           This module prints an error message.  The message is printed
'       in the error color, a beep is sounded, and the user is requested
'       to press the ESC key to continue.  The message is then erased
'       and the cursor returned to its original position.
'

SUB print.error(in.string$)

        ' **** VARIABLE DECLARATIONS
    LOCAL col%          'column cursor is on when called
    LOCAL response$     'keypress
    LOCAL row%          'row cursor is on when called


        ' *** save cursor position
    row% = CSRLIN
    col% = POS

        ' *** display error message
    PRINT
    PRINT
    PRINT TAB(%entry.indent);
    COLOR %error.color
    BEEP                                    'sound error alarm
    PRINT in.string$                        'print error message

        ' *** confirm that user has seen error message
    COLOR %menu.color
    PRINT TAB(%entry.indent);
    PRINT "Press <ESC> to continue";

    DO                                      'clear keyboard buffer
```

```
330:         response$ = INKEY$
332:     LOOP UNTIL response$=""
333:
334:     DO                                  'wait for user to press <ESC>
335:         CALL get.key(response$)
336:     LOOP UNTIL ASC(response$) = %escape.key
337:
338:         ' *** clear error message and return cursor to original position
339:     response$ = STRING$(80*(CSRLIN-row%+1), " ")
340:     LOCATE row%, col%
341:     PRINT response$;
342:     LOCATE row%, col%
343:
344: END SUB
345:
346:
347:
348:
349: '
350: '   PRINT.OPTION
351: '
352: '
353: '           This module prints a given string in the menu color and
354: '       highlights the specified option key(s) using the option color.
355: '       The string to be highlighted should be at the beginning of the
356: '       input string and separated from the rest of the string by a
357: '       pipe "|".
358: '
359: '
360:
361: SUB print.option(in.string$)
362:
363:         ' **** VARIABLE DECLARATIONS
364:   LOCAL index%          'general index variable
365:   LOCAL text$           'text to highlight
366:
367:
368:     index% = INSTR(in.string$, "|")             'get text to be highlighted
369:     text$ = LEFT$(in.string$, index%-1)
370:     in.string$ = MID$(in.string$, index%+1)
371:
372:
373:     index% = INSTR(in.string$, text$)           'find where highlighted text begins
374:
375:     COLOR %menu.color                           'print left part of string
376:     PRINT TAB(%menu.indent); MID$(in.string$, 1, index%-1);
377:
378:     COLOR %option.color                         'print highlighted text
379:     PRINT text$;
380:
381:     COLOR %menu.color                           'print right part of string
382:     PRINT MID$(in.string$, index%+LEN(text$), LEN(in.string$)-index%-LEN(text$)+1);
383:
384: END SUB
```

REFERENCES

IBM. (1984). IBM Personal computer professional graphics controller technical reference (IBM order no. 6138161). In *IBM Personal Computer Hardware Reference Library, Technical Reference, Options and Adapters, Volume 3*, IBM Corporation.

Kuhl, F.P., & Giardina, C.R. (1982). Elliptic Fourier features of a closed contour. *Computer Graphics and Image Processing*, **18**, 236-258.

Zahn, C.T., & Roskies, R.Z. (1972). Fourier descriptors for plane closed curves. *IEEE Transactions on Computers*, **C-21**, 269-281.