VOLUME IX, NUMBER 3 SEPTEMBER/OCTOBER 1987 $5.00

FORTH DIMENSIONS

MATCHPOINT

THE VISIBLE FORTH

GRIDPLOT

FIRST ANS FORTH MEETING

FORTH DIMENSIONS

EDITORIAL

Two articles this month will help you to dabble with Forth-based CAI (computer-assisted instruction): Breeden's "Matchpoint" is a bite-sized portion of a full-featured CAI engine. It deals with an interesting problem for CAI designers. Those with little experience in Forth, or with less interest in CAI, may prefer experimenting with the simpler "Drill and Practice" code by Turpin. A number of enhancements suggest themselves that would be good programming exercises for beginners.

If you are one of those who sits on the bench, listening to Forth hackers and understanding some of their theory, but with little personal experience spelunking in Forth's murkier depths, "The Visible Forth" may be a good way to take your first plunge. It's a decompiler, which is good for starters, but the more you use it, the more it reveals about what makes your Forth tick (or '). Pretty soon, you'll be hacking around and messing up control structures with the best of 'em. And probably jotting down ideas for another better `NEXT` (or `TO`, or `LOOP`, or `CASE`...).

Martin Tracy called to make sure our readers know that the ANS Forth Technical Committee is accepting technical proposals. Mail proposals to Martin (acting secretary) at FORTH, Inc., 111 N. Sepulveda Blvd., Manhattan Beach, California 90266. And look for Martin's new Forth column in *Dr. Dobb's Journal of Software Tools*' October issue.

An upcoming event of interest: this year's Forth convention will mark a ten-year anniversary, an excellent time not only to see where Forth has come from, but to discuss and learn more about where it is headed. Convention organizers have scheduled presentations by key people from Forth's history, and by those who are now shaping the language's *next* decade.

But don't wait until then to join the FIG-sponsored "RoundTable" (e.g., special-interest area) on GEnie, General Electric's information service. It features an extensive library of Forth software, real-time chats with other on-line Forth programmers, a FIG Chapters section, and an area for public messages. Like similar networks, you can call this one from most parts of the U.S. via a local phone call, and GE plans to expand its international service.

Many thanks to sysops Dennis Ruffer, Scott Squires, Gary Smith, and John Hall, for their work getting the Forth RoundTable ready for use. Let them know what you think. Many of the files are from the East Coast Forth Board, graciously donated by Jerry Shifrin as seed material. (Before you dive unprepared into the GEnie discussion of Forth standards, check Jerry's notes in this issue from the first meeting of the ANS Forth committee.)

For details of the November Forth Convention, and of GEnie's sign-up offer for FIG members, refer to the FIG ads elsewhere in this issue. And don't forget FORML — see you there!

—Marlin Ouverson
Editor

Forth Dimensions
Published by the

Forth Interest Group
Volume IX, Number 3
September/ October 1987
Editor
Marlin Ouverson
Advertising Manager
Kent Safford
Design and Production
Berglund Graphics
ISSN#0884-0822

Forth Dimensions welcomes editorial material, letters to the editor, and com-ments from its readers. No responsibility is assumed for accuracy of submissions.

Subscription to *Forth Dimensions* is included with membership in the Forth Interest Group at $30 per year ($42 overseas air). For membership, change of address, and to submit items for publication, the address is: Forth Interest Group, P.O. Box 8231, San Jose, California 95155. Administrative offices and advertising sales: 408-277-0668.

About the Forth Interest Group

The Forth Interest Group is the association of programmers, managers, and engineers who create practical, Forth-based solutions to real-world needs. Many research hardware and software designs that will advance the general state of the art. FIG provides a climate of intellectual exchange and benefits intended to assist each of its members. Publications, conferences, seminars, telecommunications, and area chapter meetings are among its activities.

"*Forth Dimensions* is published bi-monthly for $24/36 per year by the Forth Interest Group, 1330 S. Bascom Ave., Suite D, San Jose, CA 95128. Second-class postage pending at San Jose, CA 95101. POSTMASTER: Send address changes to the Forth Interest Group, P.O. Box 8231, San Jose, CA 95155."

LETTERS

Fractal's Dimension

Dear Marlin,

First, let me congratulate you on the new look of *Forth Dimensions*. The cover is particularly striking, and the style is quite effective.

I particularly enjoyed the article on fractal landscapes, which I have awaited eagerly since I saw it demonstrated at the last FORML conference. The waves were a nice touch.

Unfortunately, I could not find Figures One-a to One-d referred to in the article.

Most importantly, I would like to point out that the third paragraph of the article is somewhat misleading. A fractal is an object with a *fixed* dimension. As the article stated, most coastlines have lengths which increase as we measure them with finer and finer precision. However, the *dimension* of the coastline remains fixed.

Perhaps all will become clearer if I define a version of fractional dimension known as "capacity." Cover the fractal F with a number of balls (or disks), each of diameter d. Count the number of balls, then let d get smaller, and count the minimum number of balls of the new diameter it takes to cover F; continue in this manner. If $N(d)$ is the number of balls of diameter d it takes to cover F, then the capacity, or dimension, of the fractal may be defined as the following limit:

$$\lim_{d \to 0} \frac{\log N(d)}{-\log d}$$

As a simple example, consider the Cantor set C, which is a fractal created by removing the middle third from the unit interval [0,1], then removing the middle third (of length 1/9) from each of the remaining intervals [0,1/3] and [2/3,1], and so on. Now it takes two balls (or intervals) of diameter 1/3 to cover C, while it takes four balls of diameter 1/9 to cover C. In general, 2^n balls of diameter $1/3^n$ will cover the set. Thus, the dimension of C is log 2/log 3.

There are other notions of dimension, such as the classical Hausdorff Dimension, in which one covers the set with convex sets rather than balls, but the above definition is one of the simplest.

On another topic, I converted Mr. Koopman's excellent source code to F83. I used my own F83 assembler version of Bresenham's line-drawing algorithm for the PC, and added the ability to save and restore pictures to and from disk (also in assembler), as well as a menu with a few other options.

Thanks,

Mark Smiley
Department of Mathematics
Auburn University at Montgomery
Montgomery, Alabama 36193-0401

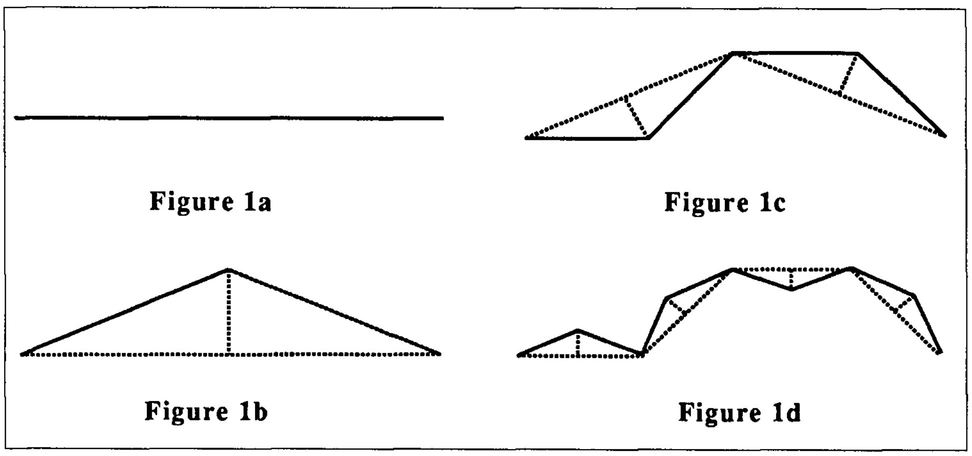


Figure 1a

Figure 1b

Figure 1c

Figure 1d

Fractal Landscapes Figures.

Brand Name Dropping

Dear Mr. Ouverson,

I am composing a list of commercially available software that is (or was at one time) written using Forth. I request anyone knowing of such products to contact me via the East Coast Forth BBS (703-442-8695) or at my address below. Also, the source of the information should be included (e.g., "Rapidfile," per Ashton-Tate's 3/87 *Technotes*). Following is my list to date:

VP-Planner
Zoomracks I & II
VALDOCS

Pan-Center
Alpine Encounter
Total Music
General Ledger
Easywriter
Expert-2
Telescan Analyzer
Micro-Aid/05
Easy3D
Quartet
Tradeshow
Delta Draw
Game Designer
Rapidfile
Neon
Starflight
Typing Tutor III
Amnesia
Back to Basics
Forthwrite
Assyst
Abundance
META
8051SIM
Chipwits
Race Car Simulator
Pro 3D
Disk Labeler

Thank you,
Michael Nemeth
10025 Locust Street
Glenn Dale, MD 20769

[Editor's note: We have no way at this time of verifying all the items in the above list, so we can make no claims for its accuracy. More information will make it better and more complete, so if you know of something to add, correct, or annotate somehow, send Michael a note. We think anything approaching a complete list would be very useful, even illuminating.]

Run DOS Files in F83

Dear Editor,

I would like to contribute a method to run F83 programs written in standard PC-DOS text files.

Like many users of Laxen and Perry's F83 Forth, I immediately looked for a way to write programs with my favorite text editor (PC-Write). Conversion of ASCII text files to Forth blocks, and vice versa, is not built into F83. Some very nice programs to do so have appeared in *Forth Dimensions* (e.g., "Screenless Forth," VIII/5).

It is possible, however, to run programs written in ASCII text files without any addition to F83. To do so, simply redirect the input stream to come from the text file.

There are two catches: In the text file, an end-of-line usually takes the form of a carriage-return/line-feed. F83 expects only a carriage-return. If you use PC-Write as an editor, then it is a simple matter to remove all line-feeds from the file. The second problem to overcome is inherent in the use of I/O redirection. At the end of the text file, F83 will wait for more input from the file. This can cause a "hanging" of the system. You can solve this by including a `SAVE-SYSTEM` and a `BYE` at the end of the text file.

Below is a simple example of how the method works. The textfile `MYPROG.TXT` contains a Forth program followed by the `SAVE-SYSTEM` and `BYE`. The resulting Forth system, `MYFORTH.COM`, contains an executable version of the program:

MYPROG.TXT, the ASCII text of the Forth program:

```
: binary ( — ) 22 base ! ;
 save-system myforth.com
bye
```

DOS commands to get an executable version of MYPROG:

```
f83 < myprog.txt
myforth
```

A good, full-screen editor for the Forth environment is still preferable, but this is a handy workaround. I hope others will benefit from it.

Yours sincerely,
Richard de Rozario
Unit 8, 6 Waverton Avenue
Waverton, Sydney 2060
Australia

F83 Gets Gregorian

Finally, Matt Wilson from Yagoona, Australia, sent his F83 version of Allen Anway's Gregorian date routine (FD IX/1), "...together with enough to get a calendar listing." Further elaborations, anyone? —Ed.

Wilson's F83 calendar screens:

1

```
\ perpetual calendar primitives FDIX1p34          MRW87JUL11

: GREGDAY   (S year month day -- ud )
   ROT ( MM DD YY/YYYY )
   DUP 100 < IF 1900 +  THEN
   ROT ( DD YYYY MM )   special, prev year
   DUP  3 < IF  12 + SWAP
                1-  SWAP  THEN
   2- 3059 100 */  ( month calc. )
   ROT  ( YYYY U DD )   32 + +  ( YYYY U )  0  ( YYYY UD )
   ROT ( UD YYYY )
   DUP 100 / NEGATE SWAP  ( UD N YYYY)
   DUP 400 / ROT + S>D ROT  ( UD D YYYY)
   36525 UM* 100 MU/MOD  ( UD D mod UD )
   ROT DROP D+ D+ ;      ( UD )

```

2

```
\ perpetual calendar primitives FDIX1p34          MRW87JUL11
: WEEKDAY    (S UD -- mod )  ( gregorian-day -- 0-6 )
( 0= Sunday,  1=Monday, 2=Tuesday, etc. )
   3 0 D+  7 MU/MOD  2DROP ;

: NTH-SUNDAY   (S year month week -- date )
   7 * 1+ DUP ( Y M D D )  >R
   GREGDAY WEEKDAY  R> SWAP - ;

: DAYS/MONTH   (S year month -- days )
   2DUP 1+ 1 GREGDAY  2SWAP 1 GREGDAY  D- DROP ;

: .WEEKDATES   (S year month day -- )
   >R  DAYS/MONTH  R>  ( lastday day )
   DUP 7 + SWAP DO    DUP I 1 ROT BETWEEN IF
       I 4 .R   ELSE    4 SPACES  THEN  LOOP   DROP  ;
```

3

```
\ perpetual calendar                              MRW87JUL11
: .DAYS    ."  Sun Mon Tue Wed Thu Fri Sat" ;

: .MONTH   (S year month -- )
   CR  2DUP . ." /" . CR  .DAYS CR    6 0 DO
   2DUP 2DUP I NTH-SUNDAY .WEEKDATES CR  LOOP  2DROP ;
: .YEAR (S year -- )
   DUP . CR 13 1 DO      I 16 .R  I 1+ 32 .R  I 2+ 32 .R CR
                  .DAYS 4 SPACES .DAYS 4 SPACES .DAYS CR
      6 0 DO   ( for each week)
          J 3 +  J DO  ( each month across page )
              DUP I 2DUP J NTH-SUNDAY .WEEKDATES 4 SPACES
              LOOP   CR   ( end of line )
          LOOP     ( end of weeks )
      3 +LOOP   ( end of three month set )
    DROP ;
```

6

```
\ perpetual calendar primitives FDIX1p34          MRW87JUL11

GREGDAY by Allen Anway
   Forth Dimensions, Volume IX, Number 1, page 34.

Gregorian Day N =  Int(365.25y) + Int(y/400) - Int(y/100)
                       + Int(30.59(m-2)) + d + 32
   where January is month 13 of previous year,
       February is month 14 of previous year.

Note that 0.59 month fraction can be any value between
     7/12 & 6/10.

Thanks to W.C. Elmore, Am. J. Phys. 44 482 (1976) (May issue)
  for the original formula
```

7

```
\ perpetual calendar primitives FDIX1p34          MRW87JUL11
WEEKDAY by Allen Anway.  W = (N+3)mod 7.

(Since there is no range checking, certain liberties can be
    taken in the calendar calculations.  They work.)
NTH-SUNDAY returns the "date" of the nth Sunday in the month.
   The "date" may be negative, or past end of month.

DAYS/MONTH calculates the number of days in the month
   by finding the difference in days between the
   first of this month and the first of the next month.
.WEEKDATES prints the dates (or spaces) for a week.
  Spaces are printed for any dates outside the valid range
   for the month, thus allowing for column alignment.
```

8

```
\ perpetual calendar                              MRW87JUL11
.DAYS for heading

.MONTH to print a given month.
   The year and month stay on the stack for the loop.

.YEAR to print a given year ( either YYYY or YY - assumes 19YY).
   Sorry for the multiple nesting of loops.  Getting the indices
    right is not difficult with bottom-up coding, but
    looks horrible when completed.
```

DRILL-AND-PRACTICE NUMBER CONVERSION

RICHARD H. TURPIN - STOCKTON, CALIFORNIA

The ease with which the base for number conversion is changed in Forth can be used to advantage in many applications. In the code listed below, this feature of Forth is at the heart of a drill-and-practice system for binary number conversion. It was used by students in an introductory digital design course. In addition to giving the students practice in number conversion, it gave me a chance to illustrate Forth to them. The application was originally written in fig-FORTH on an AIM 65 microcomputer, but has been rewritten in F83 for presentation in this article.

Lines 13 and 15 of screen 3 illustrate the basic idea used to implement the system. Variables `OUT-BASE` and `IN-BASE` hold the bases to be used for interpreting output and input data, respectively. `QUIZ` administers the practice sessions by presenting numbers using `OUT-BASE` data as base, and interpreting answers using `IN-BASE` data as base, until five correct responses have been given.

The numbers needed for the practice sessions are derived by means of a random number generator defined in screen 1, lines 10 and 11 (see Burton, "The Game of Reverse," *FD* III/5). `SET-UP` presents a number to the student, `GET-ANSWER` receives the student's response, and `TEST-IT` evaluates the response. Because the original version was administered using the AIM 65 (with its single-line display), the quiz prompts were kept simple. One change that might make the drill system more user friendly would be to display the two reference bases as reminders for the student, and a count of correct answers.

Of course, the really neat thing abut this code, typical of many Forth applications, is the ease with which variations can be generated. Screen 4 gives several more quizzes, including two for conversion between hexadecimal and binary, and one from base three to base five (I dare you to try that one!). Some examples of the program dialog are included, too. In the examples, underlined information is the program output, the rest is the student's response.

Richard H. Turpin, Ph.D., is a professor of Electrical and Computer Engineering at the University of the Pacific.

```
OPEN NUMBERS.BLK OK
Type any key, please.
 ok
BINARY.TO.DECIMAL
 1101011     107
CORRECT
10000101     133
CORRECT
 1100011     99
CORRECT
10100110     165
ANSWER IS 166
 1101101     109
CORRECT
10111100     188
CORRECT
5 CORRECT
GOOD WORK!    ok
```

```
HEX.TO.DECIMAL
       CF    3087
ANSWER IS 207
       39    57
CORRECT
       10    16
CORRECT
       62    98
CORRECT
       C3    195
CORRECT
       25    37
CORRECT
5 CORRECT
GOOD WORK!    ok
```

```
HEX.TO.BINARY
       EE    11101110
CORRECT
       A0    10100000
CORRECT
       63    01100010
ANSWER IS 1100011
       B6    10110110
CORRECT
       CE    11001110
CORRECT
       6E    01101110
CORRECT
5 CORRECT
GOOD WORK!    ok
```

```
BINARY.TO.OCTAL
 1011100     134
CORRECT
 1010000     120
CORRECT
10101100     254
CORRECT
  111000     70
CORRECT
 1001011     113
CORRECT
5 CORRECT
GOOD WORK!    ok
```

1

```
\ Drill and Practice in Number Conversion           01MAR86RHT
\    by Richard H. Turpin
\    Prof. of Elec. Engr.
\    Univ. of the Pacific
\    Stockton, CA 95211
\
VARIABLE IN-BASE
VARIABLE OUT-BASE
VARIABLE SEED
CR  .( Type any key, please. ) CR  KEY  SEED !
: RND  ( n -- n )
    SEED @ 259 * 3 + 32767 AND DUP SEED ! 32767 */ ;
: GET-A-NUMBER  ( -- n )  256 RND ;
: INPUT ( -- d,a )  PAD 20 0 FILL
    PAD 10 EXPECT  0 0 PAD  1- CONVERT  ;
-->
```

2

```
\ Drill and Practice in Number Conversion           01MAR86RHT
: "WHAT  2DROP  CR  ." WHAT?  " ;
: RESPONSE  ( - n )
   BEGIN  INPUT  C@  WHILE  "WHAT  REPEAT  DROP  ;
: GET-ANSWER  ( -- n )   IN-BASE @ BASE !  RESPONSE  ;
: SHOW-IT  ( n -- )  CR  OUT-BASE @ BASE !  8 .R  3 SPACES  ;
: CORRECT  ( c,n -- c+1 )  CR DROP 1+ ." CORRECT "  ;
: INCORRECT  ( n --)  CR  ." ANSWER IS " .   ;
-->
```

3

```
\ Drill and Practice in Number Conversion           01MAR86RHT
: TEST-IT  ( a,n -- )
   OVER =  IF  CORRECT  ELSE  INCORRECT  THEN  ;
: CONGRATS  CR  ." 5 CORRECT "  CR  ." GOOD WORK!  "  ;
: SET-UP  ( -- n )   GET-A-NUMBER  DUP  SHOW-IT  ;
: QUESTION  ( c -- c' )  SET-UP  GET-ANSWER  TEST-IT  ;
: QUIZ  0 BEGIN  QUESTION  DUP 5 =  UNTIL  DROP
   DECIMAL  CONGRATS  ;
: DECIMAL.TO.BINARY  10 OUT-BASE !   2 IN-BASE !  QUIZ  ;
: BINARY.TO.DECIMAL   2 OUT-BASE !  10 IN-BASE !  QUIZ  ;  -->
```

4

```
\ Drill and Practice in Number Conversion           01MAR86RHT
: HEX.TO.BINARY     16 OUT-BASE !   2 IN-BASE !  QUIZ  ;
: BINARY.TO.HEX      2 OUT-BASE !  16 IN-BASE !  QUIZ  ;
: BASE3.TO.BASE5     3 OUT-BASE !   5 IN-BASE !  QUIZ  ;
: BINARY.TO.OCTAL    2 OUT-BASE !   8 IN-BASE !  QUIZ  ;
: OCTAL.TO.BINARY    8 OUT-BASE !   2 IN-BASE !  QUIZ  ;
: DECIMAL.TO.HEX    10 OUT-BASE !  16 IN-BASE !  QUIZ  ;
: HEX.TO.DECIMAL    16 OUT-BASE !  10 IN-BASE !  QUIZ  ;
```

6

```
                                                    01MAR86RHT
Provides practice in conversion of numbers from one
 base to another.

Storage for base for data input interpretation.
Storage for base for data output interpretation.
Storage for random number generator seed.  Reference:  Burton,
  THE GAME OF REVERSE, Forth Dimensions, Vol. III, No. 5.
  Key input at compile time seeds random number generator.

Get a number for drill question.
Accept response from keyboard.
 Returns double no. and addr. of first nonconvertible char.
```

7

```
                                                    01MAR86RHT
Input numeric response from student.
 Reject invalid response and prompt with "WHAT?".
 (Character at address "a" from INPUT must be null character
 for good data.)  Convert to single precision number.

Input student response and interpret using IN-BASE data .

Output a number to student.
Use OUT-BASE data as base for data display.
Response for a correct answer.  DROP reference data.

Response for an incorrect answer.  Display reference data.
```

(Continued on page 36.)

the ninth annual

FORML CONFERENCE

The original technical conference
for professional Forth programmers, managers, vendors, and users.

Following Thanksgiving, November 27-29, 1987

Asilomar Conference Center
Monterey Peninsula overlooking the Pacific Ocean
Pacific Grove, California, USA

Theme: Forth and the 32-bit Computer

Computers with large address space and 32-bit architecture are now generally available at industrial and business sites. Forth has been installed and Forth applications programs are running on these computers. Graphic displays and applications are currently demanded by users. Implementation of Forth and meeting these requirements is a challenge for the Forth professional. Papers are invited that address relevant issues such as:

Large address spaces in 32-bit computers.
The graphic display, windows, & menu handling.
Relation to operating systems, other languages, & networks.
Control structures, data structures, objects, & strings.
Files, graphics, & floating point operations.
Comparison with 16-bit computers.

Papers on other Forth topics are also welcome. Mail your abstract(s) of 100 words or less. Completed papers are due November 1, 1987. For registration information call the Forth Interest Group business office at (408) 277-0668 or write to **FORML Conference.**

Asilomar is a wonderful place for a conference. It combines comfortable meeting and living accommodations with secluded forests on a Pacific Ocean beach. Registration includes deluxe rooms, all meals, and nightly wine and cheese parties.

RESERVATIONS FOR NINTH FORML CONFERENCE
Registration fees for conference attendees includes conference registration, coffee breaks, and notebook of papers submitted, and for everyone rooms Friday and Saturday, all meals from lunch Friday through lunch Sunday, wine and cheese parties Friday and Saturday nights, and use of Asilomar facilities.

Conference attendee in double room - $275 • Non-conference guest in same room - $150 • Children under 17 in same room - $100 • Infants under 2 years old in same room - free • Conference attendee in single room - $325

Send reservation requests to:

FORML Conference, Forth Interest Group, P. O. Box 8231, San Jose, CA 95155

MATCHPOINT

J. BROOKS BREEDEN - COLOMBUS, OHIO

A major advantage of computer-assisted instruction (CAI) is that lessons may be repeated as often as the learner wishes, or until some prescribed "mastery level" has been attained. One of the methods used in CAI to improve retention of material is to quiz the learner on what has been presented. Since the lesson can be repeated, to be effective a quiz should not ask the same questions in the same order every time. This paper describes a simple method for thoroughly "scrambling" a multiple-choice, matching CAI quiz.

Currently, I am developing a computer-assisted instruction tutorial on highway superelevation for my class in landscape architectural construction. In the quiz module, I included questions to identify forces acting on a vehicle in a curve by name and by formula. One question tags elements of a force vector diagram by letter, and lists element names by number; another uses the same diagram with a numbered list of formulae. Each question consists of eleven matching identifications: "What is letter a?" etc. The student enters the corresponding number of the name or formula; the program checks the response, gives feedback, shows the number of the correct response, and waits for a keypress to continue. Figure One shows a typical display.

Obviously, though, if students repeat the identical quiz several times, some will remember that "the answer for a is 7," rather than learn the correct name and formula for each element. Therefore, I wanted to randomly arrange the items presented so that on the first try, *a* might be "centrifugal force," and the next time *a* might be "weight force normal to the slope," and so forth. The problem was how to ask the questions in alphabetical order, yet randomly assign letters to the diagram components while keeping track of the correct answers.

First, I sketched the diagram on paper and listed the names of the elements. Next, I numbered the list of names and somewhat arbitrarily assigned a letter to each element on the diagram. *a* was "7. θ (theta)," *b* was "3. Force parallel to the slope," etc. The relationships originally established were as shown in Figure Two.

Suppose we shuffle the letters of the elements. If we tag the diagram in the same order, the *position* of each letter both on the display and in the list of elements will still directly correspond to its answer in the list of names. We can tag the three θ's (sevens) with three different letters, but they are still θ's. (θ by any other letter...) *Match*ing the character's position in the element list will *point* to the answer in the Name# list. *Matchpoint!*

Remember, the numbers used to identify the names and formulae are not shuffled, and the diagram does not change. Only the identifying letter tagging each element of the diagram changes. Changing only the order of letters in the element list changes both the letter used to tag each element on the displayed diagram and, because the questions are always asked in alphabetical order, the order in which the student is asked to identify any given element.

About the Code

Screens 0 through 9 contain definitions to run a facsimile of the matching questions used in the quiz. The tutorial is written in LMI's PC/Forth 3.1 and requires both EGA graphics and floating-point extensions. The definitions listed here do require LMI's EGA graphics but not the floating-point extensions; the method itself requires neither.

Screen 1 contains several words to simplify formatting, and screen 2 builds the arrays. LMI's word `,"` (comma-quote) creates an uncounted ASCII string, `A$`. One could also `C,` (C-comma) the numerical ASCII values of the first eleven lowercase letters after `CREATE`ing `A$`. `ANSWERS` holds the "correct answer" numbers corresponding to the letters in `A$`. `'CSWAP` and `SHUFFLE`, from *Thinking Forth*, Appendix D, "shuffle" `A$`'s alphabetical order using the random number routines found in *Starting Forth* (pg. 265).

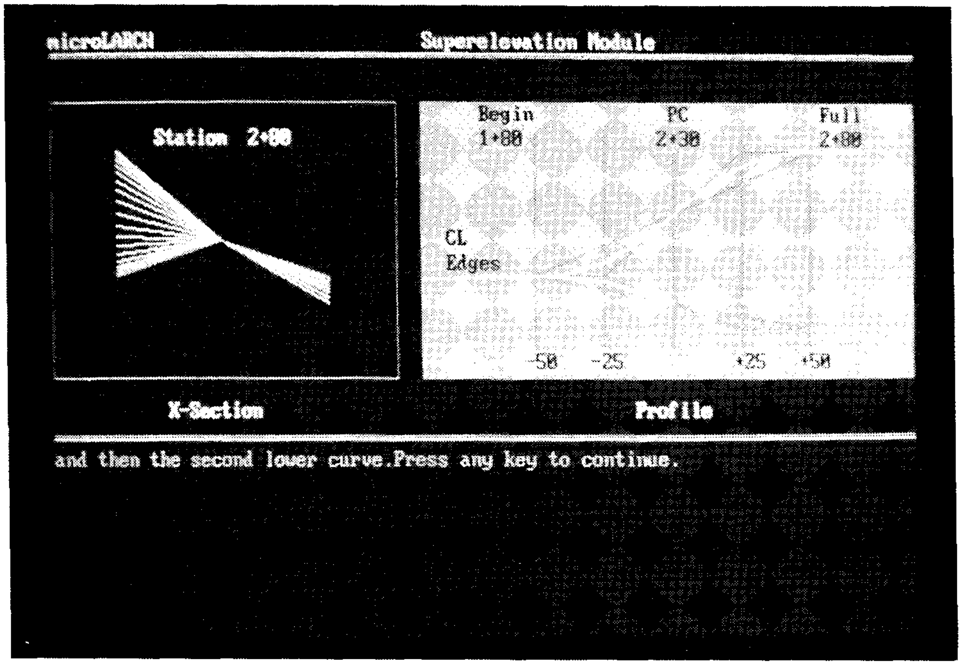
Once `A$` has been shuffled, the order of the characters is random, but each character's position in the shuffled string is still directly associated with the numeric answer stored at the corresponding position in `ANSWERS`. In other words, we don't care *what* letter is at position 3 in `A$`. The letter at position 3 in `A$` will be displayed *on the diagram* to tag the *same element* every time. The correct answer for any character tagging that element will always be 2 because 2 is stored in `ANSWERS` at position 3. Figure Three illustrates how this works.

In the shuffled `A$` in Figure Three, *a* has replaced *j* in the original `A$`. To answer the first question, "What is letter a?" we locate the position of *a* in `A$` (in bytes). The corresponding answer for that position is stored at `2*` that number of bytes from

ANSWERS, so fetching the correct number from ANSWERS is all that's required. We match (find) the character which points to the answer. *a* is located nine bytes "downstring" from A$, so the corresponding answer is located nine cells "downnumber" from ANSWERS, in this case the number 9. *b* is located where *a* was located in the original A$, so next time through the loop, the correct answer for "What is the letter b?" will be 7 (position 0 in A$, 2*, ANSWERS, +).

remember), adds ANSWER's address and fetches the correct numeric answer. It is that simple!

Screen 8 contains simple definitions for answer processing to help explain GIVEQUIZ in screen 9. GIVEQUIZ loops through ASCII a - j, incrementing the character, finding its position in A$, finding the corresponding position in ANSWERS, and evaluating the student's response. Because the loop's I is used to calculate the ASCII value of the letter of the element being asked, the questions are asked in alphabetical order.



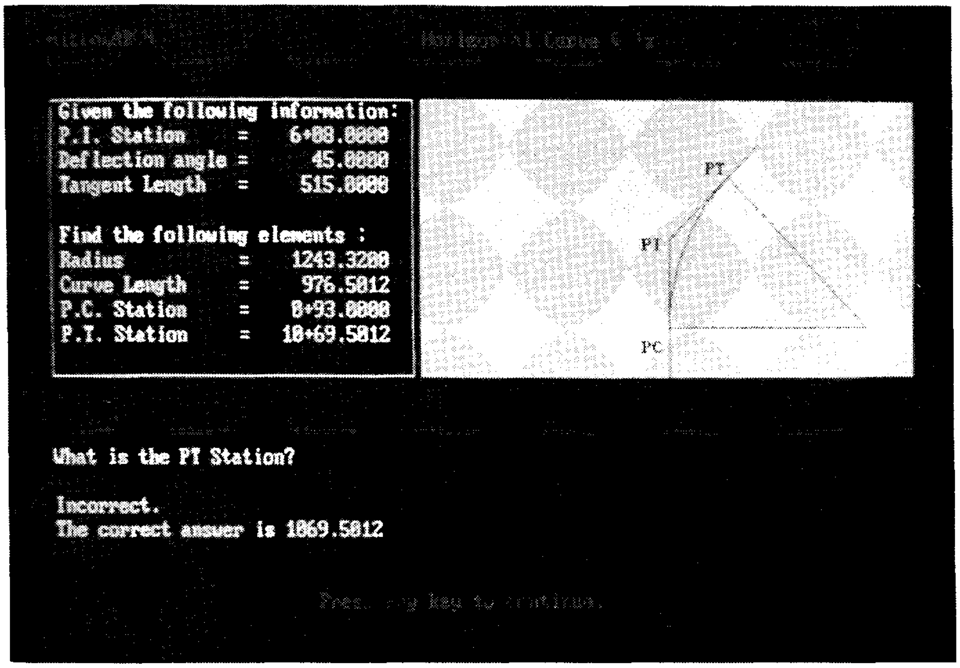
Skipping to screen 9, FINDCHAR takes the index of the loop in GIVEQUIZ and adds 97 (ASCII "a") to put the ASCII value of the *i*th letter of the alphabet on the stack. It then puts A$'s address and 11 (A$'s length) on the stack, ROTs the character to the top, and SCANs A$. SCAN (an LMI word) returns the address of the byte which contains the character and the number of bytes remaining in the string. We don't care about the remaining bytes, so we DROP them, leaving the character's address. Subtracting A$'s address leaves the offset in bytes, the position of the letter in A$. (If you don't have SCAN, see Michael Ham's "Wordwrapping Tool" (*Forth Dimensions* VII/4).

MATCHANSWER then takes the position (bytes) that FINDCHAR leaves on the stack, multiplies by two (bytes to cells,

Screens 3 through 7 contain words required to display the problem. In screen 3, the 2VARIABLE CP holds the current cursor position. DRAW draws from CP to x,y on the stack, and updates CP to the endpoint; AT resets CP to a new location.

SMALLFONT and NORMFONT use LMI's ROM BIOS call video-io to select one of the two resident EGA fonts. The font routine is not "proper," in that IBM says the call should be used only immediately following a "mode set" which clears the screen. Using the routines shown allows the use of both fonts on the screen at the same time. It may not be "proper," but in graphics mode, it works!

Screen 4 definitions draw the diagram, and TAG'EM in screen 5 tags the elements.



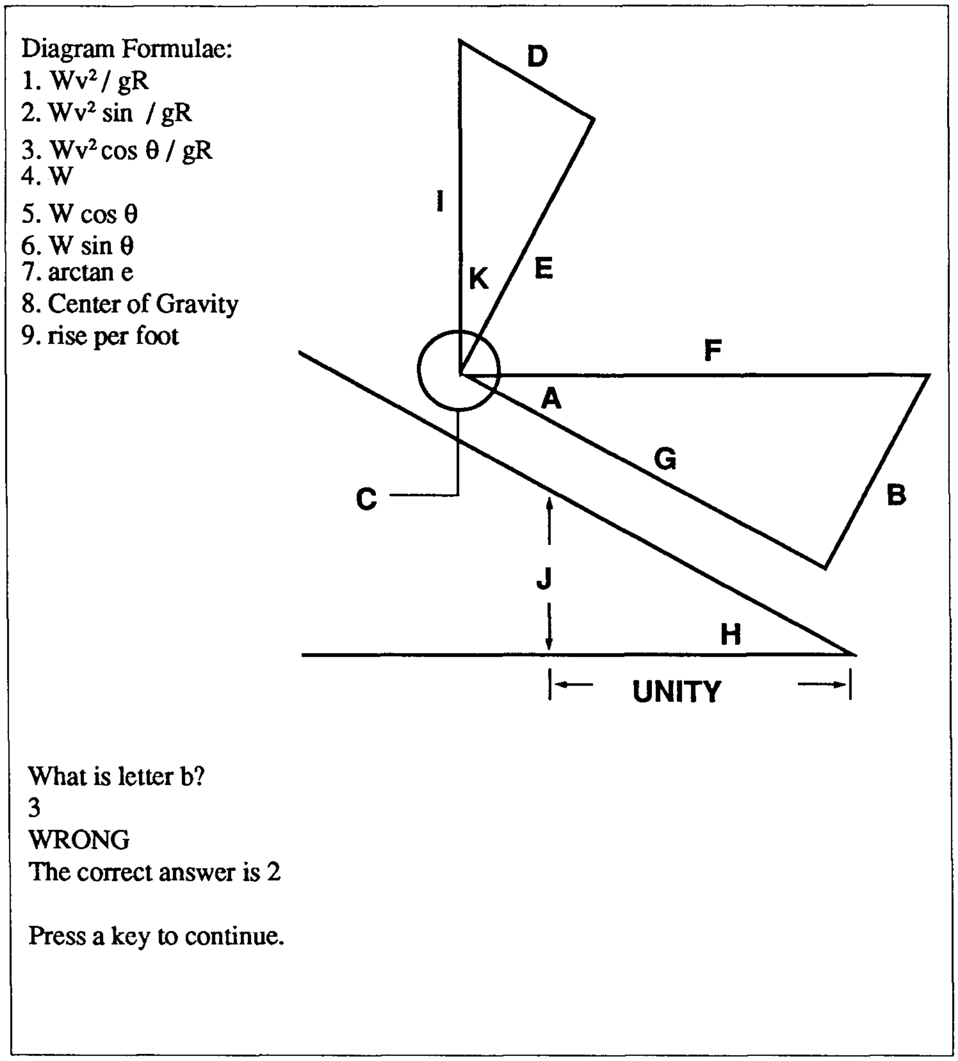


Figure One. Typical CAI display.

Screens 6 and 7 list numerical options beside the diagram.

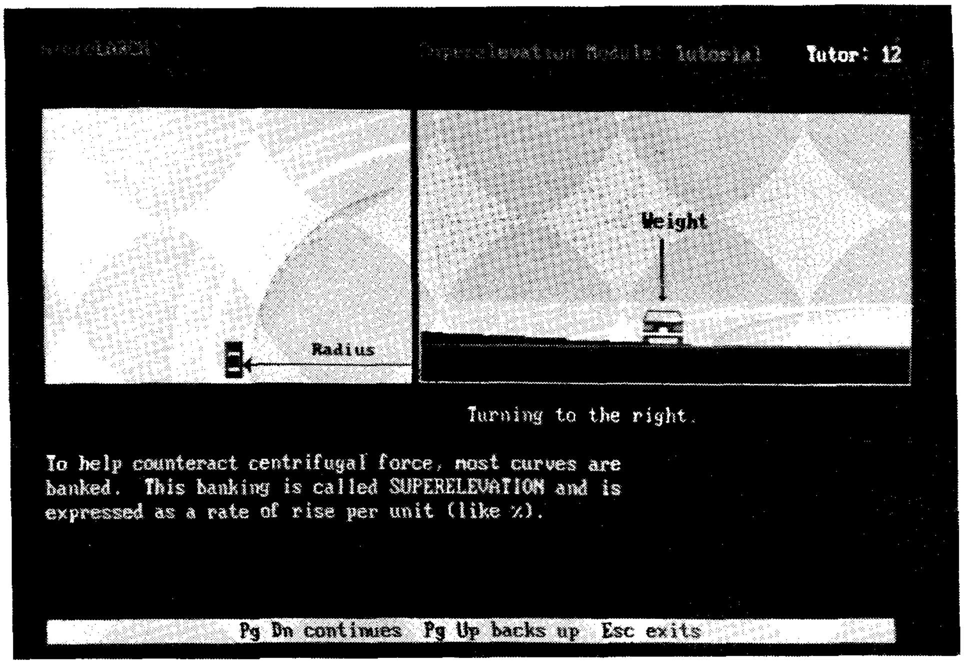
The definitions in screen 8 are simplified "cousins" of the answer-processing routines actually used. Because the rest of the quiz is mathematical problem-solving, the "real" GET-ANS uses a modified, floating-point variation of Michael Ham's "Write Like a Fox" numeric-entry routine (*FD* VI/3). CHECK-ANS chooses a random feedback string instead of just displaying "right" or "wrong," etc., but the tutorial isn't the point here. The incredibly simple technique in the example screens produces two matching problems consisting of twenty-two thoroughly scrambled questions with evaluated answers. That's Forth!

(You may add your own scoring routines...)

A Sidebar for the Interested

The superelevation tutorial which spawned this article is currently eighty-some screens of definitions which compile to a 56K LMI turnkey. This autumn, the tutorial is being used in the data-gathering phase of a doctoral dissertation, *Examination of the Cognitive Style Construct Field-Dependent/Independent as a Student Selection Criterion in Formative Evaluation.* In developing CAI, as in developing any program, user feedback is useful. The "formative evaluation" will result in two modified tutorials: one based on feedback from field-dependent students, and one based on feedback from field-independent students. All three forms of the tutorial will be used to analyze the effectiveness of feedback from students with different learning styles on the development of learning materials.

This is Brooks' second CAI-related contribution. In FD *VIII/3, he shared his technique for placing and editing the graphical elements of his presentations.*



Element:	a	b	c	d	e	f	g	h	i	j	k
Name#:	7	3	1	2	8	7	6	4	5	9	7

(Note that 7 is the correct answer for three items because the angle, θ (theta), occurs three times.)

Figure Two. Original order of elements and names.

Position:	0	1	2	3	4	5	6	7	8	9	10
Original A$:	a	b	c	d	e	f	g	h	i	j	k
ANSWERS:	7	3	1	2	8	7	6	4	5	9	7
Shuffled A$:	b	d	e	h	j	i	k	f	g	a	c

Figure Three. Shuffled A$ contents correlate to correct ANSWERS via their position.

Screen # 0

```
( Matchpoint                              jbb 12:06 10/12/86 )
( Last change:   Screen  006              jbb 12:16 10/12/86 )

                        MATCHPOINT
          A Simple Computer-Assisted Instruction
         Technique for Scrambling Matching Quizes

                            by
                     J. Brooks Breeden
                     Copyright, 1986

 This program as written, requires LMI's EGAGRAPH.COM graphics
 driver be loaded first, or you will crash.

   Written in Laboratory Microsystems, Inc.'s PC/FORTH 3.1
```

Screen # 1

```
( Formatting tools                        jbb 06:54 10/10/86 )
: OVERLAY  ;                                 ( dummy to forget)
         ( the following words simplify formatting the display)
: theta   233 EMIT ;                        ( greek letter, theta)
: SQ      253 EMIT ;                ( superscript 2 for "squared")
: V2/GR  ." v" SQ ."  / gR " ;
: TAB ( x y - )   GOTOXY ;                ( ...its so much shorter)
                            ( simulate LMI's window control for demo)
: >ENTRY-W  0 17 2DUP TAB 6 0  DO  40 SPACES CR  LOOP  TAB ;
: D-LINE   80 0 DO 205 EMIT LOOP ;                  ( double line)

VARIABLE RND
: RANDOMIZE ( - )  @TIME RND ! DROP ;  ( seed RND using clock)
-->
( RANDOMIZE seeds RND when the turnkey application boots)
```

Screen # 2

```
( Randomly shuffle string                 jbb 12:08 10/12/86 )
          ( RANDOM & CHOOSE are from "Starting Forth" p. 265.)
: RANDOM    RND @ 31421 * 6927 + 32767 AND DUP RND ! ;
: CHOOSE ( n - n)   2* RANDOM M* SWAP DROP ;

    ( LMI's "nocount" string from LMI newsletter, August, 1986)
: ," ( - )   ASCII " WORD DUP COUNT ROT SWAP DUP ALLOT CMOVE ;

CREATE A$ ," abcdefghijk"          ( compile "nocount" string)
                           ( now compile corresponding answers)
CREATE ANSWERS  7 , 3 , 1 , 2 , 8 , 7 , 6 , 4 , 5 , 9 , 7 ,

  ( 'CSWAP & SHUFFLE adapted from "Thinking Forth", pp. 268-69.)
: 'CSWAP ( a1 a2 - )   2DUP C@  SWAP C@  ROT C!  SWAP C! ;
: SHUFFLE   11 0  DO  A$ I + A$ 11 CHOOSE +  'CSWAP  LOOP ;
-->
```

Screen # 3

```
( Color/Graphics                          jbb 12:10 10/12/86 )
7 CONSTANT GRAY     1 CONSTANT BLUE
2 CONSTANT GREEN    4 CONSTANT RED
: FG    FOREGROUND ;                  ( its so much easier to type)

2VARIABLE CP                      ( coordinates of graphic cursor)
: AT ( x y - )    CP 2! ;          ( put graphic cursor "AT" CP)
: DRAW ( x y - )    2DUP CP 2@ LINE AT ;  ( draw line/update cp)

HEX         ( "illegally" select EGA resident graphics fonts)
: SMALLFONT   0 0 3 1123 video-io 4DROP ;  ( 8x8 graphics font)
: NORMFONT   0 0 2 1122 video-io 4DROP ;  ( 8x14 graphics font)
DECIMAL
-->
( LMI's  video-io  calls the ROM BIOS video service)
```

Screen # 4

```
( Draw the diagram                        jbb 12:11 10/12/86 )
: SLOPE   GRAY FG  300 180 AT  600 180 DRAW 300 60 DRAW ;
: CG-MARK   470 119 AT  470 140 DRAW  455 140 DRAW  470 112 10
   circle ;                        ( circle center of gravity)
: CF-DIAG   RED FG  470 112 AT  620 112 DRAW  588 158 DRAW
   470 112 DRAW ;             ( draw centrifugal force vectors)
: W-DIAG   GREEN FG 470  112 AT  470 32 DRAW  512 51 DRAW
   470 112 DRAW ;                        ( draw weight vectors)

: HEADER   0 0 TAB RED FG ." microLARCH" 35 0 TAB
   ." Superelevation Module: Quiz" ;
: SETUP    640X350 VMODE  CLS HEADER 0 1 TAB BLUE FG D-LINE
   0 16 TAB D-LINE ;
-->
```

Screen # 5

```
( Tag the elements                        jbb 20:24 10/09/86 )
                      ( tag elements of diagram with letters)
: TAG'EM   SHUFFLE  SMALLFONT  GRAY FG
   62 14 TAB A$     C@ EMIT  67 16 TAB A$ 1+  C@ EMIT
   68 12 TAB A$ 2+  C@ EMIT  76 18 TAB A$ 3 + C@ EMIT
   55 17 TAB A$ 4 + C@ EMIT  59 11 TAB A$ 5 + C@ EMIT
   62  4 TAB A$ 6 + C@ EMIT  57  9 TAB A$ 7 + C@ EMIT
   62 11 TAB A$ 8 + C@ EMIT  62 20 TAB A$ 9 + C@ EMIT
   69 21 TAB A$ 10 + C@ EMIT
   NORMFONT  GREEN FG
   62 13 TAB ." ;" 27 EMIT ." --" GRAY FG ." unity"
   GREEN FG ." --" 26 EMIT ." ;"  62 12 TAB 25 EMIT
   62  10 TAB 24 EMIT  >ENTRY-W ;
-->
```

(Continued on page 36.)

Dateline San Jose...
FIG Now Accessible Via Modem

Dateline San Jose

The Forth Interest Group (FIG) is up and running on GEnie, the growing modem access communication network that is part of GE Information Services. GEnie is a nationwide service that is also available in Canada and Japan, with plans for expansion into Western Europe. You can now communicate with other Forth users, and download software and commentary from the FIG sponsored Forth RoundTable on GEnie. Type "FORTH" at any main menu prompt to get to the FIG area.

The FIG area on GEnie has three main areas that are, in turn, divided into several smaller categories. The three main areas are:

- FIG Bulletin Board
- FIG Real Time Conference
- FIG Software Library

1. Bulletin Board - A message oriented information exchange with the following categories:

- Introduction to FIG Online
- Introduction to Forth
- Techniques and Tutorials
- Product Announcements
- Calendar of Events
- Applications in Forth
- Forth Dimensions Feedback
- FIG Chapter Topics
- FIG Order Line
- Forth Standards
- The Job Market
- Questions and Answers

2. Live RoundTable - Discussion area where pre-announced, interactive discussions may take place. There are three separate areas for people to gather and exchange information or chitchat.

3. File Area - A large selection of software and commentary files are available for downloading.

GEnie, RoundTable and GE Mail are trademarks of General Electric Company, U.S.A.

SPECIAL SIGN-UP FOR FIG MEMBERS

FIG members who sign up using the special FIG account number at the regular $18 one-time enrollment fee will get three free hours of time on GEnie (6 P.M. to 8 A.M.). After the initial three hours, all subsequent time is billed at the usual GEnie rate of $5 per hour during non-prime-time hours. This is one of the lowest rates around for an information service. Be warned, though, that the 8 A.M. to 6 P.M. rate is $35 per hour! So watch your online times to avoid unwanted prime time charges.

To join GEnie, use your modem to call 1-800-638-8369 with your communication parameters set at no parity, 8 data bits and 1 stop bit. Use 300 or 1200 baud setting and half duplex (echo on). Following connection, type HHH (without CR). Then, at the U# prompt, enter XJM11849,GENIE (CR) to sign on. This will put you at a menu where you may have a demonstration of GEnie, get your local telephone access number (node), and billing information. You will have the option to sign up for GEnie. Have your VISA, Mastercard, Discover, or American Express card handy or have the initial subscription fee electronically transferred from your checking account by using the number on your checks.

Next you will have to answer a few questions (name, address, etc.), and electronically agree to the terms of the service agreement. For immediate access, use a credit card. GEnie will issue a password and provide the phone number for your local GEnie access node which will allow you to use the GEnie system. You will receive your user's manual and other information shortly.

The GEnie customer service (voice) number is 1-800-638-9636 if you have any problems. Service is available from 9 A.M. to 1 A.M. weekdays and noon to 8 P.M. weekends and holidays. (Eastern Time).

fig-FORTH

THE VISIBLE FORTH

RICH FRANZEN - MIDLAND, TEXAS

We all need tools, sometimes to take things apart, sometimes to put things together. Of course, this is true of programmers just as much as it is of mechanics, cooks, and burglars. Forth makes both the design and utilization of programming tools easy, letting each one of us establish a programming environment that suits our needs and personalities.

This article will give you some tools to help you explore your Forth environment. The main word (tool) I present is called `SEE`; it is a high-level Forth decompiler. Many versions of Forth already contain a decompiler, but mine, in addition to the source code, displays the addresses for each element. This enables detailed exploration of the dictionary structure, and it gives all the information necessary to perform seat-of-the-pants patches. It can be compared in power with either a sledgehammer or a jeweler's screwdriver, depending on how you use it. Sometimes both are necessary.

You probably realize that one of the main differences between Forth and traditional languages is the number of intelligent words each language contains. Simply, every word in Forth is intelligent. It contains within itself a complete description of what it does and who it is. Compare this to language X, wherein every word is void, a mere character string, with the exception of the compiler or interpreter itself. Language X *is* this compiler or interpreter, unchanging, everlasting, dull. Since a Forth word knows itself, why not just ask it to share this information? This is what we are going to do.

My Forth decompiler, `SEE`, takes a word and divides it into all of its components. It then displays this information in an organized manner which is ideal for finding patch locations, investigating how a word works, and acquiring considerable knowledge on how Forth is put together. If you have already completed Forth 101, skip ahead to the section entitled, "The Visible and Invisible Tools" — the rest of us are going to investigate what these component parts are. Our initial breakdown of a Forth word will be into two parts, a header and a body.

The Id of Forth

The header is what gives Forth its interactive power. In a turnkey Forth application which has been scrunched down to the bare minimum necessary to accomplish the application, the header is not even necessary. What you have left, however, will not "feel" like Forth; it will be a word processor (only), or a video game (only), or a spreadsheet (only). They may, indeed, be very good word processors or video games, but one advantage of Forth is that it bypasses the "only's" given by other languages. A header is used by the outer interpreter when compiling new words into the dictionary or interpreting commands from the keyboard. It is not used during the actual execution of a word[1] and, hence, has nothing to do with the speed of execution, no matter how big the header is. The Forth header is subdivided into two parts, or "fields."

Who Am I? The first is the name field. It begins with a length/attribute byte which tells the length of the name, up to 31 characters. This leaves three bits free to tell the interpreter if the word is immediate or not, or smudged or not. Okay, that is two bits; the third bit is used by different versions of Forth in different ways. In fig-FORTH, and possibly in other versions, the high bit is always *set* ("1") for a reason I will explain momentarily. Immediately following the length byte of the name field is the name in regular ASCII text. Assuming you have neither a lower value stored in `WIDTH` nor a version of Forth with a fixed number of characters in the name field, this can be from one to 31 characters. I blinked when I said "regular ASCII text." The last character has its high-order bit set (if the name is only one character long, it is the last character). So, in fig-FORTH, the first byte of the name field has its high bit set, as does the last byte, while all bytes in between (if there are any) do not. This allows the word `TRAVERSE`, when given any address within the name field, to find either end of the name field. This is how the FIG Model handles variable-length names.

Where Am I? The second part of the header is the link field. This is just one cell (16 bits) wide. When the outer interpreter is hunting for a given word in the dictionary, it starts at the top (`LATEST` word) of the dictionary and checks if it is the word it is looking for. If not, it goes to the link field to discover where the name field of the previous word in the dictionary begins. `SMUDGE`d words are ignored. This continues until either the desired word is found or the link field contains zero, which would announce the bottom of the dictionary and an unsuccessful search. Note that the trend in newer versions of Forth is to place the link field immediately prior to the name field, a format which allows quicker dictionary searches[2]. If your version does this, a simple modification to my code can be made.

So there is the header, both name field and link field. The body is also divided into two fields, the code field and the parameter field. The body is required, even in the minimal turnkey system mentioned above. It is what makes the Forth word intelligent.

What Do I Do? The code field is one cell wide, commencing immediately after the link field. It is a pointer, and it always points to machine language code (it took this a while to get through my thick head). This code may be for colon definitions, variables, constants, `CODE` definitions, or whatever other animals may be in your system, but it is always low-level, machine-executable code. Sometimes, as in the case of `CODE` definitions, this code immediately follows the code field. Usually, however, it is a very short routine explaining how to handle the parameter(s) of the parameter field. For example, if I am a `CONSTANT`, I take the next cell after the code field and push its contents onto the stack. I am then through, my mission completed. `NEXT`.

What (or Who) Do I Do It With? The parameter field is made up of as many cells as are necessary, from zero up. In some cases, such as for constants and variables, these are parameters in the sense that other languages use the term. For the common case of colon definitions, however, we have very special parameters, a list of cells containing the code field addresses of the words used to define this word.

Hold on tight, I am going to try to confuse you. Above I said that the code field always points to machine code. However, where the code field of a given word is located, its "code field address," is not machine code. Thus, in this list created by a colon definition, none of the code field addresses point directly to low-level code. With the decompiler, you are going to see things that seem very strange until you understand this concept.

Additionally, sometimes the decompiler will display what I call "invisible words." Not many of us have actually used `0BRANCH` or `LIT` in our definitions, but when we use `IF` or a literal, they magically appear. As we think about what has to happen, the reason for these invisible words

```
scr # 2
( Miscellaneous core words and modifications)        DECIMAL
243 VARIABLE MAXSCR  ( will be 307 on non-system disk )
: LOAD    ( scr#)  DUP MAXSCR @   > 0=
    IF  LOAD  DECIMAL  ELSE  6 ERROR  ENDIF ;
: \    ( skip rest of line)
    IN @  64 /  1+ 64 *  IN ! ; IMMEDIATE
: THRU  ( n1 n2)  \ load screens n1 thru n2 inclusive
    1+ SWAP DO  I LOAD  LOOP ;
CREATE BYE   HEX 0C3 C, 0 ,  SMUDGE  DECIMAL
: CCONSTANT   <BUILDS  C,  DOES>  C@ ;
16 CCONSTANT 16                    4 CCONSTANT 4
;S

scr # 3
\ .words to aid programmer
: .D  ( n) BASE @ SWAP  DECIMAL .  BASE ! ;
: .VOC    CONTEXT @ 8 -  NFA ID. ;
: .DEF    CURRENT @ 8 -  NFA ID. ;
: 0.R  ( d cnt) <#  0 DO  #  LOOP  #>  TYPE ;  \ .R with 0 fill
: .BASE    BASE @ .D ;
: .H  ( n) BASE @ SWAP  HEX U.  BASE ! ;
: .B  ( n) BASE @ SWAP  0 16  2 BASE !  0.R SPACE  BASE ! ;
: .S  \ nondestructively print data stack
    SP@ S0 @ SWAP  2DUP = IF  1 MESSAGE  DROP DROP
      ELSE  DO  I @ .  2 +LOOP ENDIF ;
: .FRE    SP@ HERE -  .H ;
: U?   @ U. ;  \ "U-query"
;S

scr # 7
\   Non-VOCABULARY extensions
: -QUIT   ?TERMINAL IF  KEY 3 = IF  QUIT  THEN THEN ;
: BETWEEN  ( n1 n2 n3 -- f)  \ check n1 between [n2,n3]
    ROT DUP  ROT <  >R  <  R>  AND ;
: ~EMIT  \ emit only ascii between 32 and 127 else emit .
    DUP  31 128 BETWEEN IF  EMIT  ELSE  DROP ." ."  ENDIF ;
: PAD2  ( --a) PAD C/L + ;
;S

scr # 8
 \ DUMP, hex and ascii
: BUNCH  ( n1 n2) MOD 0= IF  SPACE  ENDIF ;
: (DUMP)  ( lo hi) CR
   1+ SWAP DO  I DUP  0 4 0.R  58 EMIT SPACE
      16 0 DO  DUP I 4 BUNCH  I + C@
               DUP  0 2 0.R  I PAD2 + C!  LOOP
      3 SPACES  16 0 DO  I PAD2 + C@  ~EMIT  LOOP
   CR  -QUIT  DROP  16 +LOOP ;
: HEADDUMP   CR ." addr   "  OVER  17 1 DO
      DUP 0 16 U/ DROP .  I 4 BUNCH  1+  LOOP  DROP  CR ;
: DUMP   BASE @ >R HEX  HEADDUMP  (DUMP) R> BASE ! ;
;S
```

becomes clear. For example, if our colon definition contains the literal 5783, we need a filter to say, "Hey, Inner Interpreter, the next cell is not an address of a code field; it is a number. Just let it be pushed onto the stack when we are executing." This filter is the word `LIT`. Many immediate words, such as `IF`, in addition to doing something during compilation, compile invisible words such as `0BRANCH` followed, in this case, by how many cells to skip when the value on top of the stack (during execution) is zero.

This concludes my Forth 101 lesson. Note that other people divide Forth words in different ways, sometimes using different terms. Understanding the above will help you to comprehend someone else's explanation. I hope this background information will help you to appreciate the beauty and simplicity of Forth, which you will find with `SEE`.

The Visible and Invisible Tools

Now we can start talking about the tools to make your version of Forth visible. Like almost any tool, `SEE` requires other tools to build it. A useful one, even apart from `SEE`, is a `DUMP` word that displays both hex and ASCII. Every version of Forth I have seen (except, unfortunately, the raw FIG Model) contains some sort of memory `DUMP` word, but they do not always show both hex and ASCII. Sometimes (shiver), they are merely decimal dumps. This is a "personality" word; when you call it, do you give it the starting address and a count to dump, or do you wish to give it the starting and ending addresses? The choice is entirely yours. The version I wrote uses the latter approach (I was used to it, since my ROM monitor uses this method). If you prefer the former method, merely add this redefinition on line 12 of screen 8:

```
: DUMP    ( adr count — )
  OVER +   DUMP ;
```

This version is formatted for a 64-column screen and was modelled after the D command of the excellent CP/M public-domain disk utility DUUP.

Note that some of the words used here are from screen 7. The nicest of these is `-QUIT` ("don't quit"), which normally does nothing (it doesn't quit). If it senses a

```
scr # 9
  0 \ Modification & extension to Eaker's CASE statement
  1 \   ... n CASE  ( n --)
  2 \     n1 OF  ---  ELSE  ...  n2 OF  ---  ELSE  ...
  3 \     ( or)  n3 R > IF  ---  ELSE  ...   ( any conditional test)
  4 \     [ OTHERWISE  ... ]  ( word is optional)
  5 \  ENDCASE  ...
  6
  7 : CASE   ?COMP  COMPILE >R  4 -1 ; IMMEDIATE
  8 : OF   COMPILE R  COMPILE =  [COMPILE] IF ; IMMEDIATE
  9 : OTHERWISE ; IMMEDIATE  \ strictly for appearance in source
 10 : ENDCASE   BEGIN [COMPILE] THEN  DUP -1 = UNTIL  DROP
 11    4 ?PAIRS  COMPILE R>  COMPILE DROP ; IMMEDIATE
 12 ;S
 13    If used within a DO loop, the loop index (I) will not be
 14 available within the CASE structure.
 15

scr # 16
  0 \ SEE, a high-level forth decompiler  by Rich Franzen
  1 0 VARIABLE SEE!
  2 : GETSEE!  ( -- a)  SEE! @  2+  DUP SEE! ! ;
  3 : CDMP   GETSEE! DUP  31 +  DUMP ;
  4 : SNAME  ( cfa) 2+ NFA  SPACE ID. ;
  5 : SNUM   ( n) BASE @  SWAP  HEX  0  4 0.R  BASE ! ;
  6 17 LOAD \ exceptions screens
  7 : ISEE   CR  \ initialize SEE! and print header
  8    CR  -FIND  IF  DROP DUP NFA  DUP  DUP SNUM 3 SPACES C@ .H CR
  9      DUP 1+ SNUM  6 SPACES ID. CR  LFA DUP  DUP SNUM SPACE  @
 10      DUP SNUM  ."  ( " ID. ." )" CR
 11      SEE! !  ELSE  ." Not found." QUIT  ENDIF ;
 12 : see   BEGIN  GETSEE!  DUP SNUM SPACE  @ DUP DUP  DUP SNUM
 13      DUP SEE! @ -  2 = IF  ."  code" CR  CDMP  SP! QUIT  ENDIF
 14      EXCEPTIONS IF  DUP SNAME CR  ENDIF  -QUIT
 15      0= UNTIL  DROP  CR ;   : SEE   ISEE see ;  ;S

scr # 17
  0 HEX   \ SEE exceptions                                       1
  1 : .2ND   ( n) GETSEE! @  SNUM  SPACE ;
  2 : .2TH   ( n) .2ND  SEE! @  DUP @ +  ."  (to " SNUM ." )" ;
  3 : EXCEPTIONS   CASE
  4      611 OF  ."  :" CR                    DROP   0  ELSE \ :
  5      447 OF  SNAME                             0 0  ELSE \ ;
  6      192 OF  SNAME  .2TH CR                    0  ELSE \ 0BRANCH
  7      17A OF  SNAME  .2TH CR                    0  ELSE \ BRANCH
  8      1A8 OF  SNAME  .2TH CR                    0  ELSE \ (LOOP)
  9      1E2 OF  SNAME  .2TH CR                    0  ELSE \ (+LOOP)
 10      66D OF  ."  VARIABLE "  .2ND       DROP 0 0  ELSE \ VAR
 11      653 OF  ."  CONSTANT "  .2ND       DROP 0 0  ELSE \ CONST
 12      653 OF  ."  CONSTANT "  .2ND       DROP 0 0  ELSE \ CONST
 13      67F OF  ."  USER "  GETSEE! C@  DUP .H  126 @ +
 14              @  ." = " SNUM             DROP 0 0  ELSE \ USER
 15 -->

scr # 18
  0 \ exceptions, continued                                      2
  1      A82 OF  SNAME CR  CDMP                    0 0  ELSE \ (;CODE)
  2      ACC OF  ."  does> "  GETSEE! @  .2ND .2ND
  3              .2ND CR  2- SEE! !         DROP   0  ELSE \ defword
  4      A0B OF  SNAME  GETSEE! @  SNAME CR          0  ELSE \ COMPILE
  5      156 OF  SNAME  .2ND CR                     0  ELSE \ LIT
  6      B5B OF  SNAME  GETSEE!  DUP COUNT TYPE
  7              C@ 1- SEE! +!  CR                  0  ELSE \ (.")
  8      20D OF  ."  I"                      DROP 0 0  ELSE \ pfa I
  9    ENDCASE ;
 10 ;S
 11
 12
 13
 14
 15
```

keypress, though, it pauses the listing until another key is pressed. If either the initial or the second keypress is ^C, it performs a QUIT; otherwise, the second keypress continues the display. Other keyboard monitors (than that of an Exidy Sorcerer) may react slightly differently, since my ?TERMINAL word goes directly to a special ROM routine rather than to the normal keyboard scan. If the definition does not work on your computer to your satisfaction, play with it until it does. It is a very useful word for which you will find many applications. A substitute, which will merely cause a QUIT if any key is pressed, may be made. It is not as nice, but it is still useful, and it should work with any kind of keyboard scan:

```
: -QUIT   ?TERMINAL IF
  QUIT THEN ;
```

Since I use many tool-building tools, I have included screens 2, 3, and 9. Screen 2 contains some suggestions found in Leo Brodie's *Thinking Forth*. Particularly useful is the word \, which allows a line comment (similar to ; in many assemblers). Note that the definition for LOAD automatically resets BASE to DECIMAL after the LOAD (the other stuff surrounding LOAD is peculiar to my disk configuration). The only words here necessary for SEE are 0.R (i.e., .R with zero fill) and .H (often called H., which would be inconsistent, since D. is already taken, and D. does *not* force a decimal . of the stack). This is mainly a group of words to let me know what my present system configuration is; I tend to change bases more often than a softball player. Use what you like; after all, it is your Forth. Screen 9 is an improvement to Dr. Charles Eaker's CASE statement. This is used for the different CASEs of exceptions.

Finally, I am ready to discuss the tool for which you are reading this article. The SEE tool presented here occupies three screens (16 - 18). Screen 16 should work without significant modification on any version of the FIG Model (however, on cell address machines, change the 2+ on line 2 to a 1+, and change the 2 on line 13 to a 1). There are two if's: If you choose a DUMP definition requiring address/count, on line 3 delete the words DUP and +. If your LOAD word does not automatically do so, on line 6 reset BASE to DECIMAL after loading the exceptions screens (or redefine LOAD). Only minor modifications are necessary on other versions; just do the things you have to do in converting any program. That may not be much help, but all the logic is there. I have adapted it to three systems, one of them pre-fig-FORTH, using this logic[3].

Since you are a Forth programmer, I have saved a little bit of work for you. I promise the work will be worthwhile; merely adapting the program to your system will, in itself, teach you much about your Forth. Screens 17 and 18 will require some interactive play on your part. The addresses contained therein will be correct for an unmodified 8080 fig-FORTH (mine is modified, but I kept the starting address of everything the same), but for most of you, some address changes will need to be made.

The two initial words on screen 17 make the handling of invisible words simpler and improve the display. Then comes EXCEPTIONS itself, which is merely a series of cases. If the address contained on top of the stack (when EXCEPTIONS is called) matches one of the exceptions, that code is executed and at least one zero flag is returned to see (two zero flags lets see know that it is through). If there is no match, the non-zero address (interpreted as TRUE) is returned, which causes see to treat it as if it were a normal code field address, which it will be 90% of the time. The word SNAME on screen 16 is the heart of see, assuming there are no exceptions. It merely takes a code field address from the top of the stack and prints the name of the word which has that CFA.

Still, 10% of the cases must be handled separately, so we have to adapt EXCEPTIONS for those cases, and with your addresses. The most straight-forward method here is brute force. Start out with a null version of EXCEPTIONS (: EXCEPTIONS DROP ;). Then SEE words whose definitions you already understand pretty well; like SEE HERE (the fig-FORTH definition is : HERE DP @ ;). Immediately before it starts printing garbage, an address will be displayed which will be the exception address needed. (Remember that -QUIT will allow you to break at any point by typing ^C.) On the first run of SEE HERE, this will be the address for : (which is not an ordinary definition in fig-FORTH, so in my exception I print the : directly, DROP the CFA, and return zero).

After you have made this modification to EXCEPTIONS (get rid of the DROP that was the initial definition!), run SEE HERE again. Now it will perform beautifully all the way to the ;. When it prints the subsequent garbage, it will already have printed both the address of ; and the one-character string ; (this is an ordinary fig-FORTH definition, but it needs to be an exception so see can know when to stop). Now add the ; exception per the model on line 5 of screen 17. Non-fig-FORTH models may not act normally with the ; — the way to tell is to note if a ; and CR were printed out. If they were, the ; was treated normally. If not, treat the exception in much the same way : was treated on line 4, only follow the DROP by two zeroes.

```
3086   85
3087        TITLE
308C 3078 ( SEE )
308E 0611 :
3090 0B5B (.") The
309C 0B5B (.")   Visible
30B4 0B5B (.") Forth
30C2 0447 ;S
```

Similarly, all the other exceptions may be found and added to EXCEPTIONS. You have a big advantage over me when I first did this, in that you have a listing of words to look for (to the right on screens 17 and 18). Your system probably has exceptions not noted on these two screens, however. Any defining word using ;CODE will yield exceptions and, on some systems, even defining words built from <BUILDS DOES> pairs will yield exceptions. Do not feel cheated by having to hunt down all the cases I may have missed or did not have.

Instead, remember that you are making your version of Forth visible to you; the more you learn about it, the greater its visibility will be.

There is a more orderly approach you may take in this adaptation; it will not save any time, but it will prevent gross hacking. You have a tool I did not have, the word ISEE ("Initializing SEE")[4]. For the right-most words on screens 17 and 18, merely type ISEE : or ISEE 0BRANCH. Three lines will be typed. The bottom line will be the link field, which is contained in the address to its left. The following cell after this address (two bytes, or one cell, depending on your system) will be the code field, which will usually be the exception address you were looking for. In the cases where it is not, such as :, do an ISEE of a colon definition (in this case), as in ISEE HERE. Then do a DUMP using the left-most address on the top line as your starting address, and about 64 higher than that for your ending address.

You will discover two exceptions upon examination of this dump, for : and for ;. The way to tell which method you will need (at least in a fig-FORTH model) is to see if my exception pattern contains the word SNAME. If it does, the first method may be used; if not, use the latter. Note that the latter method will always work, but may give you more information that you will immediately understand.

Finally, let us look at some actual SEE decompilations from my system. I have them fully commented in Figures One and Two. I tried to choose samples of many of the things you will run into. The comments are in a different typeface than the actual decompilation, and this method may be used to get a two-dimensional understanding of the way a given word works. In each sample, three fields per line result from the decompilation. The first field is the address of what follows on that line. In some cases, such as compiled strings or many of the invisible words, this is just the first address; listing all the addresses would be useless, since they follow linearly. The second field is the contents of this address, usually the CFA of another word. The third field is just the name of this word pointed to, but may be a compiled string or some other excepted animal.

Note that this does not look like a source definition is expected to look. It is not supposed to. I organized it to contain all information the word knows about itself, more than was included in the source definition. IF has become 0BRANCH, ELSE has become BRANCH, and THEN and BEGIN have disappeared entirely. There will be other surprises as well, but I promised you a visible Forth, one showing what your Forth is, not what you imagined it to be. You will discover it is better than you imagined.

I will comment briefly on CODE definitions. My listings merely print a DUMP when machine code is sensed. While a Z80 (or 6502 or 8086 or PDP-11) disassembler is quite practical, it is not a tool I have found necessary to build. Most machine code used in Forth is quite short and easy to disassemble by hand, when desired. It is your Forth; if you want this tool, build it. It can be patched into SEE by replacing your disassembler word with my word CDMP (Code Dump). Then share it with the rest of us — it is our Forth, too.

Endnotes

1. There are rare exceptions. I have seen a Forth calendar containing words for the twelve months, that uses the text within the headers to print out the name of the desired month. A Forth programmer can use any part of Forth in any manner desired.

2. Consider that the vast majority of string comparisons during a search return FALSE

(Text continued on page 37.)

```
1946   86                        Length byte (true length found by ANDing with 1F)
1947        MAXSCR               ASCII name (the max. screen number on my drive)
194D 193B ( TASK )               Link field
194F 066D VARIABLE 00F3          Parameter field  (for formatting purposes, the
                                   value is included on the same line)
```

```
06B4   83
06B5        C/L                  Characters per Line
06B8 06AB ( BL )
06BA 0653 CONSTANT 0040          Note that all numbers are shown as positive,
                                   16-bit, hexadecimal characters
```

```
07AB   84
07AC        BASE
07B0 07A0 ( STATE )
07B2 067F USER 26 = 0010         the EXCEPTION for USER automatically looks up
                                   the actual value, which had an index of 26 hex
```

```
30F5   C6                        VOCABULARY definition, built with <BUILDS ... DOES>
30F6        EDITOR                 Note what DOES> does.  It compiles addresses into
30FC 30D4 ( LINE )                 the dictionary, the first of which is where
30FE 0ACC does> A081 3613 2175     execution will be transferred.  It also pushes
0FE2 07F4 2+                       the location of the second address of the compiled
0FE4 0790 CONTEXT                  addresses onto the stack for utilization by the
0FE6 05CE !                        executable code.  A perfectly remarkable GOTO in
0FE8 0447 ;S                       disguise!
```

```
07EF   82                        Note how a CODE definition may be sensed (the 2
07F0        2+                     difference between Ø7F4 and Ø7F6).  The machine
07F2 07E2 ( 1+ )                   code ends at location Ø7FB.  Try to decompile
07F4 07F6 code                     HERE by hand.  Then get busy and type in "SEE"!

addr   6 7 8 9  A B C D  E F 0 1  2 3 4 5

07F6:  E12323C3 44018448 4552C5EF 07110645   .##.D..HER.....E
0806:  07970547 0485414C 4C4FD4FC 07110645   ...G..ALLO.....E
```

Figure One. Miscellaneous forms. Note the three columns per decompilation: Location, Contents of Location, and Description.

Forth Interest Group Presents

Ninth Annual

FORTH NATIONAL CONVENTION

November 13-14, 1987

Red Lion Inn
2050 Gateway Place
San Jose, California 95110

FIG's 10th Anniversary Celebration

EXHIBITS

CONFERENCE PROGRAM

THE EVOLUTION OF FORTH
Past - Present - Future

The founders, writers, producers, and evaluators of Forth look at:
Forth in development • Forth at work • Forth in the future

TECHNICAL SESSIONS
Covering the latest in Forth technology

Conference & Exhibit Hours

Fri. Nov.13	12 noon - 6 pm
Sat. Nov. 14	9 am - 5 pm

Hotel Reservations
Red Lion Inn, San Jose, California
800 547-8010 or 408 279-0600
Request Forth Interest Group Rates.

Preregistration $20 • Registration at the door $25 • Banquet Saturday 7 p.m. $35 (with keynote speaker)

Preregistrations @ $20 _____ Banquet Tickets @ $35 _____ Total check to FIG (US funds) $________________
Name ____
Company ____
Address ____
City______________________State ______ZIP___________Country ________________
Telephone (_______)______________________________________

Return to: Forth Interest Group • P. O. Box 8629 • San Jose, CA 95155

```
1BD5   85
1BD6       3LIST                        ...but what if I really, really want to begin with
1BDB 1BA5 ( INDEX )                       screen 16?
1BDD 0611 :
1BDF 0556 DUP
1BE1 06A7 3
1BE3 04C9 +
1BE5 0549 SWAP
1BE7 01EF (DO)
1BE9 0310 CR
1BEB 020B I
1BED 1B68 LIST
1BEF 01A8 (LOOP) FFF8  (to 1BE9)  "jumping" EXCEPTIONS will automatically compute
1BF3 0310 CR                          and show where they "jump" to
1BF5 0156 LIT 000F
1BF9 1270 MESSAGE
1BFB 1B9B PAGE
1BFD 0447 ;S
```

```
1BFF   85
1C00       TRIAD                        Ok, sometimes I'll do it the "standard" way.
1C05 1BD5 ( 3LIST )                     Note that 1BD5 is the very first byte of 3LIST
1C07 0611 :
1C09 06A7 3
1C0B 11D4 /
1C0D 06A7 3
1C0F 11B5 *
1C11 1BDD 3LIST                         Also note this link to 3LIST.  The 1BDD does not point
1C13 0447 ;S                              to machine code, but the contents of 1BDD do.
```

```
07FC   84
07FD       HERE                         provided as a favor to those who tried to decompile
0801 07EF ( 2+ )                          HERE from the DUMP in FIGURE 1.
0803 0611 :
0805 0745 DP
0807 0597 @
0809 0447 ;S
```

Figure Two. Colon-Definition Examples

```
2021   84
2022        ?BUF      What's in the disk buffers?
2026 2009 ( SCR>BLK )
2028 0611 :
202A 1CAE GRON        GRaphics ON
202C 0310 CR
202E 07B2 BASE
2030 0597 @
2032 0A56 HEX
2034 1317 #BUFF       # of disk BUFFers (24 in my system)
2036 068F 0
2038 01EF (DO)        control loop ----------------------------
203A 020B I
203C 06DE B/BUF       Bytes per disk BUFfer (256 in my system)
203E 07F4 2+
2040 07F4 2+          quicker than ( LIT ) 4 +
2042 11B5 *
2044 06C6 FIRST       address of FIRST disk buffer
2046 04C9 +
2048 0597 @
204A 0556 DUP
204C 068F 0
204E 0156 LIT 0006
2052 18D8 D.R
2054 0156 LIT 00EF    right arrow character
2058 02E0 EMIT
205A 0156 LIT 7FFF
205E 03C3 AND
2060 08D0 -DUP
2062 0192 0BRANCH 0012  (to 2076)  IF (is buffer empty?)
2066 1FFB BLK>SCR     convert block # to screen #
2068 0A6C DECIMAL
206A 068F 0
206C 06A7 3
206E 1A36 0.R         display screen # as decimal
2070 0A56 HEX
2072 017A BRANCH 0008  (to 207C)
2076 0B5B (.") vzv    ELSE (print 3 graphic characters--
207C 020B I                 which printer neutered out)
207E 07E7 1+
2080 0156 LIT 0006    do a CR after every sixth buffer
2084 11E4 MOD
2086 04A3 0=
2088 0192 0BRANCH 0004  (to 208E)
208C 0310 CR
208E 01A8 (LOOP) FFAA  (to 203A)   end of control loop --------
2092 07B2 BASE
2094 05CE !
2096 1C9D GROFF       GRaphics OFF
2098 0447 ;S
```

Figure Three. 2-Dimensional Word Analysis

AUGUST 1987

ANS FORTH MEETING NOTES

JERRY SHIFRIN - MCLEAN, VIRGINIA

These notes about the first meeting of the ANS Forth Technical Committee (August 3-4, 1987 at CBEMA Headquarters in Washington, DC) are not minutes of the meeting, nor are they part of the ANS Forth Technical Committee's official documentation. They represent my personal impressions only. The reason I mention this is that during the course of the meeting, it became clear there are serious liability considerations for anyone publishing anything which someone could interpret as anything approaching a standard. Therefore, I must caution everyone that this IS NOT a standards document. With that out of the way, I'll try to give you a summary of the events.

The meeting began at 9:00 a.m. on Monday and was attended by many of the leaders of the Forth Community, including a number of folks from the Forth Standards Team (FST). Elizabeth Rather (as the "convenor") was acting chairperson, and Ray Duncan volunteered to be acting secretary for the initial meeting. Elizabeth made some opening remarks on the maturation of Forth and the need for greater acceptance. She described the scope of this effort as mainly oriented towards describing common practice, neither attempting to fix Forth nor using the standard as an instrument for change.

The agenda was approved without dissent.

There followed a discussion on international involvement. The X3 representative suggested that early involvement would lengthen the process. It turns out that there is a requirement for an international liaison, but it wasn't clear at what point that would become important. Chuck Moore felt strongly that international involvement was important. I don't recall that any decision was reached on this.

Following this was a discussion on validation suites. The original scope proposal specifically excluded them from this group's activity. The membership voted not to exclude the possibility of such development.

Cathy Kachurik of CBEMA presented a tutorial on the X3 structure and process.

Charlie Keane proposed adopting the Forth-83 Standard as a "BASIS" document. As I understood it, the basis document becomes the working document for all activity of the technical committee (TC). That is, all changes, deletions, additions, etc. are proposed as updates to this document. There was a discussion about whether to restrict this document to Chapter 12 (the Required Word Set). This was defeated 15-1. A motion to adopt chapters 1-16 carried unanimously. This excluded only the appendices (uncontrolled reference words, experimental proposals, charter, membership, and proposal/comment forms).

I put up a motion to include floating point as part of the scope of work. This was defeated by a vote of 4-12. A motion was approved (14-3) to eliminate the time-frame constraints on standard language extensions. Previously, this had indicated that language extensions could not be considered until the approval of the ANS Forth standard.

Two additions to the Scope of Work document were approved: that the TC will review existing and proposed programming language standards; and to consider the impact of the standard on current and anticipated hardware technology. Another change to this document was to change the number of users required for a Forth system in order to be considered for non-compliance with the Forth-83 standard; this number was reduced from 1000 to 200 users of a particular Forth implementation. The Scope of Work document (X3J14/87-002) was then approved as amended.

The TC Subcommittees document (X3J14/87-004) was approved with minor changes. There are four subcommittees: Documentation, Research, Logistics, and Technical.

The Plan of Work document (X3J14/87-003) was then approved with a few changes. It was agreed to remove specific topic areas from the meeting plan in order to allow the work to proceed faster if possible. Chuck Moore got a motion passed to require that at least one meeting be held in San Francisco. The membership then agreed to hold the next meeting in San Francisco, but could not find anyone willing to host that meeting. Elizabeth Rather then volunteered to host the meeting in Southern California, and that was agreed on. I put up a motion to co-schedule and co-locate the ANS Forth meetings with the FORML and Rochester conferences. This was defeated 4-10.

Next was the Call for Officers. The X3 Secretariat appoints the officers from a list of volunteers. The following people volunteered at the meeting:

Chair: Charlie Keane, Bill Dress, Larry Forsley

Vice Chair: Bill Dress, Ray Duncan

Secretary (appt'd.by chair):Martin Tracy

International Representative: Larry Forsley
Vocabulary Representative (appointed by the chair): Ted Dickens
Documentation Editor (appointed by the chair): Ron Braithwaite

Applications for these positions may be accepted until August 31st. Each requires a letter of intent and qualification, along with a letter from your employer indicating that they understand the amount of time needed for taking on these assignments.

Greg Bailey and Don Colburn then proposed that the technical subcommittee agree to mark up the standards document, indicating areas to be deleted, modified, added, and areas of deviation from accepted practice. This was agreed 13-3.

The group then agreed 9-4 to hold the next meeting on November 11-12, 1987 in Southern California at FORTH, Inc.

Elizabeth named acting chairs for each of the subcommittees: Ted Dickens, documentation; Gary Betts, logistics; Guy Kelly, research; Greg Bailey, technical.

I passed out documentation and gave a brief description of the ANSForth bulletin board on MCI Mail.

There was then a review of all action items and the TC adjourned. This was immediately followed by the convening of the Technical Subcommittee (TSC).

Martin Tracy volunteered to serve as acting TSC secretary. There was a lengthy discussion on the proper name for this subcommittee (I'm using TSC in these notes, but that may not be accurate) and its voting membership requirements. At issue was whether it was a formal subcommittee which would carry additional documentation requirements. As I recall, no conclusion was reached on this. We did get the impression that in order to be a voting member of the TSC, you had to be a voting member of the TC.

The TSC then drew together a list of goals: identify a kernel of highly compatible words, decide a strategy for layering and extensions, amass information on the TC desires and needs, and define a mechanism for handling proposals.

I ran out of steam around this point and stopped taking notes, but most of the remaining discussion was on the proposal process, voting membership, and plans for the next meeting.

Commentary

While the preceding describes events to the best of my recollection and note-taking abilities, I thought I'd add a few opinions and observations of my own:

First, I think this effort is off to a great start. The membership includes an excellent and reasonably well-balanced group of vendors, users, and other interested folks. I believe most of the early objections to this effort have been resolved by the make-up of the TC. Additionally, there was a clearly cooperative spirit among the attendees.

It was very clear to me that the TC was determined to pursue an open organization. Several discussions were concerned with how to publicize our procedures and encourage participation. This, along with CBEMA's requirements for "due process" will, I think, result in an excellent document.

Unfortunately, it seems there is still an IEEE cloud hanging over this effort. I thought a compromise had been reached, but apparently there are still a few people pursuing the IEEE Forth alternative. We'll have to wait and see what happens.

Chuck Moore, in spite of his avowed opposition to a Forth standard, was extremely cooperative. My impression is that he was mainly concerned with having wide participation and not shutting off the possibility of new Forth development. (He also mentioned that he was working on a new Forth compiler.)

Don Colburn seemed to feel that we could put out a draft document much earlier than planned and was surprised at the idea that there would be any difficulty in reaching a consensus.

I felt that Elizabeth did an excellent job of chairing the meeting, but suppose she was wise in not volunteering as the permanent Chair. This way, we avoid even the appearance of a FORTH, Inc.-dominated effort.

I have a couple of concerns about the course of this project. Most difficult for me to reconcile is the notion of a standard documenting common usage among the major Forth implementations. In some cases, this may cause a reversion of some language features back to the way they were before the 83 standard. For example, FORTH, Inc. never changed its definition of `LEAVE` to correspond with the 83-standard; i.e., it does not immediately leave the loop. One could, therefore, argue that the 83 standard `LEAVE` is not in common usage. Thus, it seems to me that the ANS standard might either leave its effect undefined or else omit it entirely. Worse, I think, would be to revert its meaning back to the 79 standard.

My other main concern is with the minimalist approach. I guess it's the only sensible way of getting this out in a reasonable amount of time, but I worry that most proposals will simply be put aside with the note that they're outside the documented scope of the ANS Forth effort.

On the positive side, I think this group has enough talent and dedication to complete a superb standard in a reasonable amount of time. I offer my personal thanks to everyone involved for providing two days of stimulating discussion. A special tip of my Forth beanie goes to Elizabeth Rather and Martin Tracy for doing the bulk of the work in pulling this activity together.

Other Notes of Interest

Don and Chris Colburn were kind enough to invite everyone over to their house Monday evening for pizza and pool (swimming, that is). It was very pleasant and provided the opportunity for people to get to know each other a bit better. Don took us down for a tour of his workshop; it looked like a Mac farm. Don demonstrated the Mac II running several animated graphics tasks under MacForth in separate windows. Very nice.

I was very happy to have Martin Tracy and Guy Kelly stay at my house, but regret the short time available for Forth talk. Guy demonstrated his new, implementation-independent Forth editor. It seemed very powerful — it could work with screen files or native blocks. In addition, he provided three ways for moving stuff around — cut and paste, a line stack (push and pop a line at a time), and a "barrel" (push stuff into the barrel, and select from it in any order). I believe Guy will be offering this for sale.

He includes drivers for several Forths, including F83, MVP, Uniforth, and LMI.

Martin, as usual, has numerous irons in the fire, including his polyFORTH implementation for Digital Signal Processing (DSP) chips, his upcoming *Dr. Dobb's* article and column, along with work on ZenForth, and his continuing involvement with the Forth Model Library. Martin mentioned that Wil Baden had started helping him on the Zen project. Martin stopped by the Potomac FIG meeting (on his way to the airport) and talked about several of his projects.

Miscellaneous Notes

X3 has raised its membership fees: $200 for one principal membership (includes one alternate), $150 for observer and each additional alternate. The X3 Secretariat may be reached at: CBEMA, 311 First St. N.W., Suite 500, Washington DC 20001, 202-737-8888.

Jerry Shifrin is employed by MCI and is the sysop for the East Coast Forth Board (703-442-8695).

MCI MAIL'S ANS FORTH BBS

MCI Telecommunications is sponsoring a bulletin board on MCI Mail in support of the ANSI Forth standards activity known as X3J14. This board will contain agendas, proposals, minutes of meetings, and related information. If you are interested in the development of the ANS Forth standard, this is the place to see what's going on.

ANS Forth Bulletin Board

ANSForth is the main heading for several types of message areas. From the main "Command:" prompt, type VIEW ANSFORTH to see all currently active areas. The following are currently available:

1. General: general information on X3J14; membership, documentation, officers, etc.
2. Agenda: agendas for X3J14 meetings.
3. Minutes: minutes of previous meetings.
4. Proposals: proposals for consideration by X3J14.
5. Comments: comments on active proposals.
6. Misc.: uncategorized messages.

In general, bulletin board messages will only remain available for 90 days (this is an MCI Mail constraint). Archived versions of older messages will be available for downloading from the East Coast Forth Board.

Viewing the Bulletin Board

To access the ANSForth Bulletin Board type

VIEW ANSFORTH <*subarea-name*>

For example, type

VIEW ANSFORTH AGENDA

to see the upcoming agenda.

If you don't know the particular subarea's name, type at least ANSFORTH, then choose the subarea you want from the list of matching names. You will then see a "View:" prompt; type one of the following commands:

SCAN	To display a scan table of items on the board, showing the date posted, subject, and the size of the item.
READ	To display all items or those selected by scan number, with page breaks and a pause between items.
PRINT	To display all items or those selected by scan number, with no page breaks or pauses between items.
LEAVE	To exit the board and return to mail mode.
EXIT	To log off MCI Mail.

Posting Messages on the Board

Unlike many other bulletin boards, MCI Mail only allows the bulletin board owner (me) to post messages. Therefore, in order to get a message posted for public view, you must send it to me to be forwarded.

The best way to send me a message is via MCI Mail. I check this mailbox daily, and forward messages to the bulletin board with just a few keystrokes. To send me a message on MCI Mail, enter the CREATE command and at the "TO:" prompt enter "Gerald A. Shifrin". Alternatively, you may address a message to my MCI Mail id: 299-4103.

The second best way to get a message posted is to leave it on the East Coast Forth Board at 703-442-8695, addressed to SYSOP with instructions for it to be posted on the ANS Forth Bulletin Board. Similarly, you may upload a file (also with appropriate instructions) to me. This will get your message posted within a day or two.

If you don't telecommunicate but want to get a message posted, send me a floppy disk in IBM PC format containing the file(s) to be posted. Mail this to: Jerry Shifrin, 6212 Loch Raven Dr., McLean, VA 22101.

I'm sorry, but I won't be able to return your disk unless you include a self-addressed, stamped mailer. Please do not send written material to be posted. I don't have a document scanner and I'm not a great typist.

Prices

The annual mailbox fee is $18, payable upon registration and billed on the anniversary date of service. This allows you to read messages sent to you. There are additional charges for sending messages and access fees; these are described below. The initial $18 registration fee will be credited against any charges you have in the first two months.

It costs $.45 to send an instant message of up to 500 characters; $1 for up to 7500 characters, and another $1 for each subsequent 7500 characters. There is no connect-time charge if you call through a local MCI Mail number. There is a $.05/minute charge if you access MCI Mail via Tymnet.

—*JS*

[Editor's note: You may find the standards-related news and discussion on FIG's GEnie conference (see announcement elsewhere), and on the East Coast Forth Board, timely and complete enough for most purposes.]

GRIDPLOT

GENE THOMAS - LITTLE ROCK, ARKANSAS

When I began learning Forth, I was particularly excited about finally being able to operate in high resolution on the TI without having to wrestle with assembly language. But I immediately ran into a snag. The transcendentals (sin, cos, etc.) compete with the bit-map color table for work space! No matter that I wasn't using any color commands — a messy screen resulted. How was I to draw a simple circle without sines and cosines? I know that I'm not the first to tackle this problem, and probably not the first to solve it, but here goes.

My first effort involved the use of a sin table. It worked. Then I saw a copy of *FD* IV/1 with the trig derivation screen by J. Bumgarner. With some slight modifications and a couple of additions, it became a part of my program. A reduction of memory overhead, and one less screen resulted. Some major changes in the graphics commands followed and `GRIDPLOT` was born.

The accompanying screens are reasonably well documented, so I won't discuss each word in detail. `HUE` sets the drawing color against a black background. `XSET` and `YSET` add your X and Y positions to `XC` and `YC` (constants describing the center of the screen) to effect X,Y plotting. `CLEAN` erases the bit-mapped screen, accounting for the current mode, `GRAPHICS2`, `SPLIT`, or `SPLIT2`. `?BREAK` checks for `FUNCTION CLEAR`, allowing you to halt a graphic in progress.

A more descriptive discussion of `CIRCLE` will help in understanding the graphic words. The sin of the angle of a rotating radius at each step gives the X position. In a like manner, the cos produces the Y position. These successive X,Y coordinates are plotted, forming the perimeter of the circle. `XSET` and `YSET` shift these coordinates to the correct quadrant of the X,Y grid. (See Figure One.)

A complete understanding of `TLINE` is necessary, both because of the resident word `CLINE`, and because how it works is not immediately evident. `CLINE` writes the tiny characters to the bit-mapped screen. `CLINE` expects to find on the stack the address of the text to write, the character count, and the line number to write to. The line numbers are rows zero to 23, as if in the 32-column mode. `CLINE` always begins writing at column zero. The only way to move the text over is to pad it with 32s preceding the first character. That is a major flaw when you want to write on a screen containing graphics. Those blanks will erase anything in their way.

`TLINE` is a redefinition of `CLINE`. I didn't redefine `CLINE` itself because `CLINE` is still useful, and requires less effort on our part to use when it is appropriate.

Both `TLINE` and `CLINE` call `SMASH`, a resident `CODE`d word which formats the tiny characters and puts them into a line buffer, `LB`. `SMASH` leaves on the stack the information expected by `VMBW` (Video Multiple Byte Write), the `LB` address, the `VDP` address to write to (column and row, to us), and the character count.

In `TLINE`, the `VDP` address left by `SMASH` is `DROP`ed and replaced by the `VDP` address that is the column and row we want to write to. That address is computed by `TPUT` by adding the necessary number of addresses to skip, to the beginning of the bit-map pattern descriptor table — which starts at hex 2000 (8192), and is stored in the variable `PUT`. The text is stored in `TBUF`, beginning at `TBUF+1`. The count is stored at `TBUF` (i.e., *cnt* `TBUF C!`).

Use the keyboard method (`TGET`) to find just where you want to print your text, then edit the necessary commands onto your Forth screen. See the notes on screen 29 for examples of how to do it. As a reminder, `TGET` requires two returns.

Screen 31 contains a demonstration which should help in getting started. Screen 30 contains two formulas that draw some very satisfying graphics. Writing `HYPO` and `EPI` will be valuable practice. Then add the `SCREEN_DUMP` from *FD* VI/6 and you will have a toy well worth playing with!

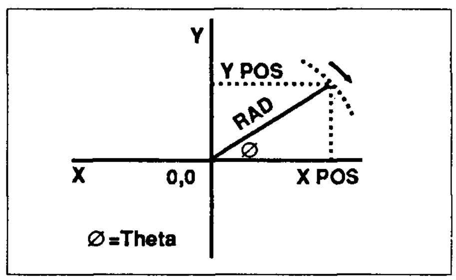


Figure One. Theta is rotated through 360 degrees in the DO LOOP, where I = theta. On each increment of the loop, the X,Y positions are plotted.

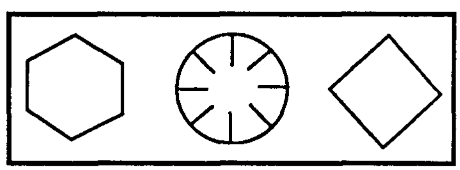


Figure Two. Sample screen dump.

```
SCR #19
( GRIDPLOT [ver 2.0] Loading screen. GT Apr 85 ) BASE->R DECIMAL
  TEXT   CLS  0 2 GOTOXY  ."      GRIDPLOT ver 2.0"      CR
                         EMPTY-BUFFERS
."          C.A.F.I.G."                              CR CR
." Scrn  Contents"                                   CR
."  ____________________________"                    CR
." 20 -  sin & cos functions"                20 LOAD  CR
." 21 -  variables and utilities"            21 LOAD  CR
." 22 -  utilities continued"                22 LOAD  CR
." 23 -  circle command"                     23 LOAD  CR
." 24 -  rule & point commands"              24 LOAD  CR
." 25 -  spokes command"                     25 LOAD  CR
." 26 -  polygon command"                    26 LOAD  CR
." 27 -  box command"                        27 LOAD  CR
." 28 -  tline (bit-map print)"              28 LOAD  CR CR
R->BASE   ." Command?"     QUIT
```

```
SCR #20
( TRIG by derivation[MOD]: 4th DEMINSIONS 4/1,    J.Bumgarner )
BASE->R  DECIMAL
10000 CONSTANT 10K       ( scale factor )
: LF  SWAP / MINUS  10K */ 10K + ; ( repititious formula )
: ?MIRROR DUP 90 > IF 180 SWAP - THEN ;
: REDUCE 360 MOD DUP 0< IF 360 + THEN DUP 180 < IF ?MIRROR
        ELSE 180 - ?MIRROR MINUS THEN ; ( establish sector )
: SIN REDUCE 17453 100 */  ( convert degrees to radians )
     DUP DUP 10K */ >R 10K 72 R LF 42 R LF 20 R LF 6 R LF
     10K */ R> DROP ;  ( calculate sin )
: COS 360 MOD 90 SWAP - SIN ; ( COS[X] = SIN[X+90] )
: SIN*  ( n angle -- n*sin[angle] )  SIN  M* 10K M/ SWAP
       4999 > IF 1+ THEN ;      ( rounded up if rem => .5 )
: COS*  360 MOD 90 SWAP - SIN* ; ( plotable X,Y positions )
R->BASE  ;S
                 Modified by GT, Apr 85
```

```
SCR #21
( GRIDPLOT COMMANDS, GT Apr 85 )      BASE->R   HEX
                  ( Variables, Constants )
80 CONSTANT XC  60 CONSTANT YC  ( center of screen constants )
 0 VARIABLE XP   0 VARIABLE YP  ( x and y positions )
 0 VARIABLE RAD  ( radius value )  0 VARIABLE INC ( increment )
 0 VARIABLE XP'  0 VARIABLE YP' ( TO equivalent of XP, YP )
 0 VARIABLE WID  0 VARIABLE DEP ( box variables )
                      ( Utilities )
: HUE  ( c --  ;p 1 cyan, 2 red, 3 yellow, 4 magenta, 5 green )
     CASE  1 OF  71  ENDOF   2 OF 61  ENDOF
           3 OF  A1  ENDOF   4 OF D1  ENDOF
           5 OF  C1  ENDOF   6 OF F1  ENDOF ( white )
     ENDCASE  DCOLOR ! ;     ( default is white )
: XSET  XC XP @ + + ;   ( X and Y offsets for plotting on )
: YSET  YC YP @ + + ;                         ( X,Y grid )
R->BASE
```

```
SCR #22
( GRIDPLOT COMMANDS, GT Apr 85 )  BASE->R  DECIMAL
               ( Utilities, cont'd )
: CLEAN  ( -- erase bit-map screen )
    PAD 64 32 FILL      ( ensure only blanks )
     VDPMDE @ CASE      ( get bit-map mode )
( graphics2 )    4 OF  24 0  ENDOF
(  split    )    5 OF  16 0  ENDOF
(  split2   )    6 OF  24 4  ENDOF
                ENDCASE
    DO  PAD 64 I CLINE  LOOP ;  ( erase each line )

: ?BREAK  ( ;p check for break [fctn clr] if in graphics2 then )
          ( go to split [to make keyboaad text visable] )
   ?TERMINAL DUP DUP  IF VDPMDE @ 4 = IF SPLIT THEN  IF CLEAN
    QUIT THEN THEN ;
R->BASE
```

```
SCR #23
( CIRCLE; GRIDPLOT  GT Apr 85 )    BASE->R    DECIMAL
: CIRC  ( [variables] circle  -- )
       361 1 DO  RAD @ I SIN* XSET  ( calculate column )
                 RAD @ I COS* YSET  ( calculate row )
                  DOT   ?BREAK      ( plot row,col; fctn clr? )
            INC @ +LOOP ;           ( plot only every INC deg )
: CIRCLE  ( xp yp rad --  ;p xp=0, yp=0 is center of screen )
       DUP 35 > IF 1 ELSE DUP 15 < IF 3 ELSE 2  ( set inc )
        THEN THEN  INC !
       RAD !  YP !  XP !   CIRC ;
R->BASE  ;S   There are some INC's and DO limits which in
combination produce smoother circles at sone radii.  Limited
resolution of 256 X 192 tends to make ragged circles.
X,Y quadrant:  upper lt  lower lt  upper rt  lower rt  center
          XP:    neg       neg       pos       pos       0
          YP:    neg       pos       neg       pos       0
```

```
SCR #24
( RULE, POINT; GRIDPLOT  GT Apr 85 )  BASE->R
: RULE   ( xp yp  xp' yp' --  ;p draw line using X,Y coord. )
         >R >R >R   XC +  R>  YC +          ( col, row from )
         R> XC +  R> YC +                     ( col, row to )
         LINE ;                                 ( draw line )

: POINT  ( xp yp --  ;p plot dot using X,Y coord. )
         >R XC +  R> YC +  DOT ;  ( 0 0 POINT is scrn center )

R->BASE  ;S


RULE draws a line from/to X,Y coordinates.  POINT plots a dot
at X,Y coordinate.  LINE and DOT commands may still be used as
usual to draw a line from col,row to col,row, and dot to plot a
pixel on at col,.row.
```

```
SCR #25
( SPOKES: GRIDPLOT  GT Apr 85 )   BASE->R   DECIMAL

: SPO    ( [variables] spokes -- )
        361 0 DO  RAD @ I SIN* XSET      ( X end position )
                  RAD @ I COS* YSET      ( Y end position )
                  XP @  XC + YP @ YC +   ( origin of spokes )
                  LINE  ?BREAK           ( draw origin to X,Y )
              XP' @  +LOOP ;             ( repeat rep times )
: SPOKES  ( xp yp len rep --  ;p plot rep spokes len from xp,yp)
          360 SWAP / XP' !  ( XP' used to store repititions )
          RAD !  ( store len )
          YP !  XP !  ( store origin of spokes )
          SPO ;
R->BASE  ;S
SPOKES plots spokes (as on a wheel) given a center, length of
spokes and number of spokes.

SCR #26
( POLY; GRIDPLOT, GT Apr 85 )  BASE->R  DECIMAL
: POL  ( [variables] poly --  ;p draw polygon )
        361 INC @ DO  RAD @ I SIN* XSET  ( X end position from )
                  RAD @ I COS* YSET      ( Y end position from )
                   RAD @ I XP' @ +  SIN* XSET  ( X position to )
                   RAD @ I XP' @ +  COS* YSET  ( Y position to )
                    LINE  XP' @              ( draw side )
                  +LOOP ;    ( repeat 360/#sides times )

: POLY    ( xp yp rad #side -- ;p draw polygon )
          360 SWAP / DUP  XP' !  INC !  ( store #side and inc )
          RAD !  ( store radius )
          YP !  XP !  ( store center position of polygon )
          POL ;
R->BASE   ;S        POLY plots a polygon by connecting the ends
of spokes, but without drawing the spokes.

SCR #27
 ( BOX PLOT,  GT Apr 85 )     BASE->R  DECIMAL
: RESET  XP' @ XP !  YP' @ YP ! ;     ( reset var to init value)
: XP+  XP @ WID @ +  XP !  0 XSET ;  ( calculaate corners )
: YP+  YP @ DEP @ +  YP '  0 YSET ;
: BX  ( [VARIABLES] box -- ) YP @ YP' ! XP @ XP' ! ( save var's)
   ( col,row fm)  ( col,row to)
  0 XSET  0 YSET   XP+ 0  YSET   LINE  RESET ( draw line & )
  0 XSET  YP+      XP+ 0  YSET   LINE  RESET ( restore var's )
  0 XSET  0 YSET   0 XSET YP+    LINE  RESET
  XP+     0 YSET   OVER   YP+    LINE ;

: BOX  ( xp yp wid dep -- ;p ul corner, wid accross, dep down )
    DEP !  WID !  YP !  XP !  BX ;
R->BASE
```

```
SCR #28
( TLINE: GRIDPLOT  GT Apr 85 ) BASE->R  DECIMAL
0 VARIABLE TBUF 34 ALLOT   ( buf for 64 char of TLINE text )
                           ( PLUS char count. )
0 VARIABLE PUT  ( PUT holds the VDP address to write to. the )
                ( address is converted by TPUT from col, row. )
: TCNT  TBUF 1+ 64 0 DO  DUP I + C@ 0= IF  I LEAVE  THEN
        ( count from 1st char of text to first null, put the )
        ( count on the stack and LEAVE the loop. )
          LOOP SWAP  1-    C! ;   ( store char count at TBUF )
: TPUT  256 * SWAP  4 *  +  8192 +  PUT ! ;  ( col, row -- )
        ( calculate VDPaddr to begin writing at, store in PUT. )
R->BASE -->

SCR #29
( TLINE: GRIDPLOT  GT Apr 85 ) BASE->R  DECIMAL
: TGET  TBUF 1+  64 EXPECT  TCNT ;  ( -- <enter> text <enter> )
        ( store text beginning at addr TBUF plus 1. )
: TLINE  LB 100 ERASE SMASH  >R DROP ( drop addr from dummy arg)
  ( and replace it with) PUT @   R> VMBW ;  ( write on screen )
  ( LB is a resident linebuffer used by SMASH. )
  ( SMASH requires a line #, thus the dummy argument. )
: WRITE  TBUF COUNT 1-  0 ( dummy argument)  TLINE ;  ( -- )
R->BASE  ;S  To print from screen use the string-store word,
!" and put the text at TBUF 1+, then put count at TBUF, give
the col, row to TPUT:  TBUF 1+ !" TEXT" cnt TBUF C! col row TPUT
Then execute WRITE. To print from keyboard use TGET,TPUT,WRITE.
ALWAYS MAKE CHAR COUNT EVEN, PAD WITH A BLANK IF NECESSARY: AND
START ON AN EVEN NUMBERED COLUMN. ROWS ARE 0 THRU 23.

SCR #30
;S  GRIDPLOT - some interesting formulas for graphics.  The
formulas are given in algebraic form.  It is left to you to
put them into post-fix notation.
Epicycloid:  X = (A+B)*COS[theta] - A*COS(A+B/A*theta)
             Y = (A+B)*SIN[theta] - A*SIN(A+B/A*theta)
Hypocycloid: X = (A-B)*COS[theta] + B*COS(A-B/B*theta)
             Y = (A-B)*SIN[theta] - B*SIN(A-B/B*theta)
Theta is the same rotating angle used to describe a circle.
A and B are the respective radii of a large and small circle,
one rolling around the other.  Imagine the one rolling has a
pencil attached and that the pencil is tracing a line.  This is
the graphic produced by the above formulas.
Epicycloid=small wheel rolling around the ouside of the larger.
Hypocycloid=small rolling around the inside of the larger.
GT  Apr 85
```

```
SCR #31
( GRIDPLOT DEMO ) BASE->R DECIMAL GRAPHICS2 4 HUE
: DEMO  0 0 25 CIRCLE  ( centered 25 pixel radius circle )
        -70 0 25 6 POLY  ( 6 side polygon, -70 X axis location )
        70 0 25 4 POLY   ( 4 side poly is diamond )
        0 0 24 8 SPOKES  ( put 8 spokes in circle )
        UNDRAW 0 0 10 8 SPOKES  ( erase part of spokes )  DRAW
        -100 -35 200 70 BOX  ( to center box, double abs of #'s)
        ( write text to bit-mapped screen )
        TBUF 1+  !" PRESS ANY KEY "  ( pad for even char cnt )
        14 TBUF C!  ( store char cnt )
        24  ( even #'d col )  20  ( row )  TPUT
        WRITE  ( write the text to the screen )
        KEY  ( wait for keypress )  SPLIT  ( go to split mode )
        ." Command? "  ( print a prompt, 32 col screen text )
        QUIT ;
R->BASE         DEMO  ( execute DEMO immediatly )
```

ADVERTISERS INDEX

(Continued from page 16.)

Screen # 6

```
( Question 1 options                    jbb 12:16 10/12/86 )
: .Q1OPTIONS   GREEN FG  0 3 TAB
   ." Diagram Elements:"  CR CR  GRAY FG
   ."  1.  Centrifugal Force" CR
   ."  2.  Force Normal" CR
   ."  3.  Force Parallel" CR
   ."  4.  Weight" CR
   ."  5.  Weight Normal" CR
   ."  6.  Weight Parallel" CR
   ."  7.  " theta ."  (Theta)" CR
   ."  8.  Center of Gravity" CR
   ."  9.  Superelevation Rate (e)"  >ENTRY-W ;

: MATCH   SETUP  SLOPE CG-MARK CF-DIAG W-DIAG TAG'EM ;
: MATCH1   MATCH .Q1OPTIONS ;
-->
```

Screen # 7

```
( Question 2 options                    jbb 12:16 10/12/86 )
: .Q2OPTIONS   GREEN FG  0 3 TAB
   ." Diagram Formulae:"  CR CR  GRAY FG
   ."  1.  W" V2/GR  CR
   ."  2.  Wv" 253 EMIT ."  sin " theta ."  / gR "  CR
   ."  3.  Wv" 253 EMIT ."  cos " theta ."  / gR "  CR
   ."  4.  W" CR
   ."  5.  W cos " theta  CR
   ."  6.  W sin " theta  CR
   ."  7.  arctan e" CR
   ."  8.  Center of Gravity" CR
   ."  9.  rise per foot   "  >ENTRY-W ;

: MATCH2  MATCH .Q2OPTIONS ;
-->
```

Screen # 8

```
( Answer processing                     jbb 12:15 10/12/86 )
VARIABLE ANS                        ( holds correct answer)
: QUESTION ( - )   ." What is letter " ;
: GET-ANS ( - n)   #IN ;
: ANSWER ( n - )   ANS ! ;
: CHECK-ANS ( n - )  ANS @ - IF ." WRONG" ELSE ." RIGHT" THEN ;
: SHOW-ANS ( - )  CR ." The correct answer is " ANS @ . ;
: WAIT ( - )  ." Press a key to continue."  KEY DROP ;
: EVALUATE ( - )  GET-ANS CR CHECK-ANS SHOW-ANS CR CR WAIT ;
-->
Note: these are "down & dirty" quick demo versions of answer
processing routines.  There are no keyboard error traps or
variations in feedback.  They serve only to illustrate the
functions for the quiz demo.
```

Screen # 9

```
( Quiz                                  jbb 12:13 10/12/86 )
                        ( find character's position in A$)
: FINDCHAR ( i - a)   97 + A$ 11  ROT  SCAN DROP A$ - ;

                     ( match position to answer in ANSWERS)
: MATCHANSWER ( a - n)   2* ANSWERS +  @  ANSWER ;

   ( Loop "a" to "k" finding answer, asking ?, & evaluating)
: GIVEQUIZ ( - )   11 0
   DO  I FINDCHAR MATCHANSWER >ENTRY-W  QUESTION I 97 + EMIT
     ." ?" CR  EVALUATE  LOOP ;

: Q1  MATCH1 GIVEQUIZ ;
: Q2  MATCH2 GIVEQUIZ ;
: QUIZ   Q1 Q2 ;
```

(Continued from page 10.)

```
   8
                                                 01MAR86RHT
Test answer by comparing to reference data.  If correct
 response, increment question count.  If incorrect, show
 correct answer but don't increment count.
Congratulate for good work in completing conversion quiz.

Obtain a number and display it to student.  Keep a copy for
 subsequent evaluation of response.
Present a number in output base.  Accept answer in input base.
 If correct c'=c+1, else c'=c.
Define a general quiz administrator to present five
 numbers for conversion.

Define a quiz for decimal-to-binary conversion by settng
 data output base to 10 and data input base to 2.
A binary-to-decimal quiz.
```

```
   9
                                                 01MAR86RHT
A hexadecimal-to-binary quiz.

Binary-to-hexadecimal.

How about base 3-to-base 5?

Binary-to-octal.

Octal-to-binary.

Decimal-to-hexadecimal.

Hexadecimal-to-decimal.
```

(Continued form page 22.)
and that this is usually known after comparing just the first character. With the link field first, the search routine can access the link immediately. Otherwise (except on constant-length name field systems), it has to AND the length byte with 31, then ADD this to the previous link field to determine the location of the current link field.

3. 6502 fig-FORTH on a Commodore 64, 8080 fig-FORTH on an Exidy Sorcerer, and an ancient, non-standard PDP-11 version of Forth, so old `EMIT` was still called `SPIT`.

4. In my hacking manner, I only recently divided `SEE` into the components `ISEE` and `see` — it had been just one big word. The wisdom of the breakdown became apparent after reading *Thinking Forth*, and after noting that every so often I needed to `SEE` a word starting somewhere after the header. I may yet begin to plan.

Rich Franzen works as an image processor/programmer for Satellite Exploration Consultants, Inc., where he uses AIMS, a very powerful image processing system that is written in an old, PDP-11 version of Forth.

FIG CHAPTERS

U.S.A.

• ALABAMA
Huntsville FIG Chapter
Tom Konantz (205) 881-6483

• ALASKA
Kodiak Area Chapter
Horace Simmons (907) 486-5049

• ARIZONA
Phoenix Chapter
4th Thurs., 7:30 p.m.
Dennis L. Wilson (602) 956-7578
Tucson Chapter
2nd & 4th Sun., 2 p.m.
Flexible Hybrid Systems
2030 E. Broadway #206
John C. Mead (602) 323-9763

• ARKANSAS
Central Arkansas Chapter
Little Rock
2nd Sat., 2 p.m. &
4th Wed., 7 p.m.
Jungkind Photo, 12th & Main
Gary Smith (501) 227-7817

• CALIFORNIA
Los Angeles Chapter
4th Sat., 10 a.m.
Hawthorne Public Library
12700 S. Grevillea Ave.
Phillip Wasson (213) 649-1428
Monterey/Salinas Chapter
Bud Devins (408) 633-3253
Orange County Chapter
4th Wed., 7 p.m.
Fullerton Savings
Huntington Beach
Noshir Jesung (714) 842-3032
San Diego Chapter
Thursdays, 12 noon
Guy Kelly (619) 450-0553
Sacramento Chapter
4th Wed., 7 p.m.
1798-59th St., Room A
Tom Ghormley (916) 444-7775
Silicon Valley Chapter
4th Sat., 10 a.m.
H-P, Cupertino
George Shaw (415) 276-5953
Stockton Chapter
Doug Dillon (209) 931-2448

• COLORADO
Denver Chapter
1st Mon., 7 p.m.
Steven Sams (303) 477-5955

• CONNECTICUT
Central Connecticut Chapter
Charles Krajewski (203) 344-9996

• FLORIDA
Orlando Chapter
Every other Wed., 8 p.m.
Herman B. Gibson (305) 855-4790
Southeast Florida Chapter
Coconut Grove area
John Forsberg (305) 252-0108
Tampa Bay Chapter
1st Wed., 7:30 p.m.
Terry McNay (813) 725-1245

• GEORGIA
Atlanta Chapter
3rd Tues.,6:30 p.m
Western Sizzlen, Doraville
Nick Hennenfent (404) 393-3010

• ILLINOIS
Cache Forth Chapter
Oak Park
Clyde W. Phillips, Jr.
(312) 386-3147
Central Illinois Chapter
Urbana
Sidney Bowhill (217) 333-4150
Rockwell Chicago Chapter
Gerard Kusiolek (312) 885-8092

• INDIANA
Central Indiana Chapter
3rd Sat., 10 a.m.
John Oglesby (317) 353-3929
Fort Wayne Chapter
2nd Tues., 7 p.m.
I/P Univ. Campus, B71 Neff Hall
Blair MacDermid (219) 749-2042

• IOWA
Iowa City Chapter
4th Tues.
Engineering Bldg., Rm. 2128
University of Iowa
Robert Benedict (319) 337-7853
Central Iowa FIG Chapter
1st Tues., 7:30 p.m.
Iowa State Univ., 214 Comp. Sci.
Rodrick Eldridge (515) 294-5659
Fairfield FIG Chapter
4th day, 8:15 p.m.
Gurdy Leete (515) 472-7077

• KANSAS
Wichita Chapter (FIGPAC)
3rd Wed., 7 p.m.
Wilbur E. Walker Co.,
532 Market
Arne Flones (316) 267-8852

• MASSACHUSETTS
Boston Chapter
3rd Wed., 7 p.m.
Honeywell
300 Concord, Billerica
Gary Chanson (617) 527-7206

• MICHIGAN
Detroit/Ann Arbor area
4th Thurs.
Tom Chrapkiewicz (313) 322-7862

• MINNESOTA
MNFIG Chapter
Minneapolis
Even Month, 1st Mon., 7:30 p.m.
Odd Month, 1st Sat., 9:30 a.m.
Vincent Hall, Univ. of MN
Fred Olson (612) 588-9532

• MISSOURI
Kansas City Chapter
4th Tues., 7 p.m.
Midwest Research Institute
MAG Conference Center
Linus Orth (913) 236-9189
St. Louis Chapter
1st Tues., 7 p.m.
Thornhill Branch Library
Contact Robert Washam
91 Weis Dr.
Ellisville, MO 63011

• NEW JERSEY
New Jersey Chapter
Rutgers Univ., Piscataway
Nicholas Lordi (201) 338-9363

• NEW MEXICO
Albuquerque Chapter
1st Thurs., 7:30 p.m.
Physics & Astronomy Bldg.
Univ. of New Mexico
Jon Bryan (505) 298-3292

• NEW YORK
FIG, New York
2nd Wed., 7:45 p.m.
Manhattan
Ron Martinez (212) 866-1157
Rochester Chapter
4th Sat., 1 p.m.
Monroe Comm. College
Bldg. 7, Rm. 102
Frank Lanzafame (716) 235-0168
Syracuse Chapter
3rd Wed., 7 p.m.
Henry J. Fay (315) 446-4600

• NORTH CAROLINA
Raleigh Chapter
Frank Bridges (919) 552-1357

• OHIO
Akron Chapter
3rd Mon., 7 p.m.
McDowell Library
Thomas Franks (216) 336-3167
Athens Chapter
Isreal Urieli (614) 594-3731
Cleveland Chapter
4th Tues., 7 p.m.
Chagrin Falls Library
Gary Bergstrom (216) 247-2492
Dayton Chapter
2nd Tues. & 4th Wed., 6:30 p.m.
CFC. 11 W. Monument Ave., #612
Gary Granger (513) 257-6984

• OKLAHOMA
Central Oklahoma Chapter
3rd Wed., 7:30 p.m.
Health Tech. Bldg., OSU Tech.
Contact Larry Somers
2410 N.W. 49th
Oklahoma City, OK 73112

• OREGON
Greater Oregon Chapter
Beaverton

2nd Sat., 1 p.m.
Tektronix Industrial Park,
Bldg. 50
Tom Almy (503) 692-2811
Willamette Valley Chapter
4th Tues., 7 p.m.
Linn-Benton Comm. College
Pann McCuaig (503) 752-5113

• PENNSYLVANIA
Philadelphia Chapter
4th Sat., 10 a.m.
Drexel University, Stratton Hall
Melanie Hoag (215) 895-2628

• TENNESSEE
East Tennessee Chapter
Oak Ridge
2nd Tues., 7:30 p.m.
Sci. Appl. Int'l. Corp., 8th Fl.
800 Oak Ridge Turnpike,
Richard Secrist (615) 483-7242

• TEXAS
Austin Chapter
Contact Matt Lawrence
P.O. Box 180409
Austin, TX 78718
Dallas/Ft. Worth
Metroplex Chapter
4th Thurs., 7 p.m.
Chuck Durrett (214) 245-1064
Houston Chapter
1st Mon., 7 p.m.
Univ. of St. Thomas
Russel Harris (713) 461-1618
Periman Basin Chapter
Odessa
Carl Bryson (915) 337-8994

• UTAH
North Orem FIG Chapter
Contact Ron Tanner
748 N. 1340 W.
Orem, UT 84057

• VERMONT
Vermont Chapter
Vergennes
3rd Mon., 7:30 p.m.
Vergennes Union High School
Rm. 210, Monkton Rd.
Don VanSyckel (802) 388-6698

• VIRGINIA
First Forth of Hampton Roads
William Edmonds (804) 898-4099
Potomac Chapter
Arlington
2nd Tues., 7 p.m.
Lee Center
Lee Highway at Lexington St.
Joel Shprentz (703) 860-9260
Richmond Forth Group
2nd Wed., 7 p.m.
154 Business School
Univ. of Richmond
Donald A. Full (804) 739-3623

• WISCONSIN
Lake Superior FIG Chapter
2nd Fri., 7:30 p.m.
Main 195, UW-Superior
Allen Anway (715) 394-8360
MAD Apple Chapter
Contact Bill Horton
502 Atlas Ave.
Madison, WI 53714
Milwaukee Area Chapter
Donald Kimes (414) 377-0708

INTERNATIONAL

• AUSTRALIA
Melbourne Chapter
1st Fri., 8 p.m.
Contact Lance Collins
65 Martin Road
Glen Iris, Victoria 3146
03/29-2600
Sydney Chapter
2nd Fri., 7 p.m.
John Goodsell Bldg., Rm. LG19
Univ. of New South Wales
Contact Peter Tregeagle
10 Binda Rd., Yowie Bay
02/524-7490

• BELGIUM
Belgium Chapter
4th Wed., 20:00h
Contact Luk Van Loock
Lariksdreff 20
2120 Schoten
03/658-6343
Southern Belgium Chapter
Contact Jean-Marc Bertinchamps
Rue N. Monnom, 2
B-6290 Nalinnes
071/213858

• CANADA
Northern Alberta Chapter
4th Sat., 1 p.m.
N. Alta. Inst. of Tech.
Tony Van Muyden (403) 962-2203
Nova Scotia Chapter
Halifax
Howard Harawitz (902) 477-3665
Southern Ontario Chapter
Quarterly, 1st Sat., 2 p.m.
Genl. Sci. Bldg., Rm. 212
McMaster University
Dr. N. Solntseff (416) 525-9140 ext. 3
Toronto Chapter
Contact John Clark Smith
P.O. Box 230, Station H
Toronto, ON M4C 5J2
Vancouver Chapter
Don Vanderweele (604) 941-4073

• COLOMBIA
Colombia Chapter
Contact Luis Javier Parra B.
Aptdo. Aereo 100394
Bogota 214-0345

• DENMARK
Forth Interesse Gruppe Denmark
Copenhagen
Erik Oestergaard, 1-520494

• ENGLAND
Forth Interest Group- U.K.
London
1st Thurs., 7 p.m.
Polytechnic of South Bank
Rm. 408
Borough Rd.
Contact D.J. Neale
58 Woodland Way
Morden, Surry SM4 4DS

• FRANCE
French Language Chapter
Contact Jean-Daniel Dodin
77 Rue du Cagire
31100 Toulouse
(16-61)44.03.06
FIG des Alpes Chapter
Annely
Georges Seibel, 50 57 0280

• GERMANY
Hamburg FIG Chapter
4th Sat., 1500h
Contact Horst-Gunter Lynsche
Common Interface Alpha
Schanzenstrasse 27
2000 Hamburg 6

• HOLLAND
Holland Chapter
Contact Adriaan van Roosmalen
Heusden Houtsestraat 134
4817 We Breda
31 76 713104

• IRELAND
Irish Chapter
Contact Hugh Dobbs
Newton School
Waterford
051/75757 or 051/74124

• ITALY
FIG Italia
Contact Marco Tausel
Via Gerolamo Forni 48
20161 Milano
02/435249

• JAPAN
Japan Chapter
Contact Toshi Inoue
Dept. of Mineral Dev. Eng.
University of Tokyo
7-3-1 Hongo, Bunkyo 113
812-2111 ext. 7073

• NORWAY
Bergen Chapter
Kjell Birger Faeraas, 47-518-7784

• REPUBLIC OF CHINA (R.O.C.)
Contact Ching-Tang Tzeng
P.O. Box 28
Lung-Tan, Taiwan 325

• SWEDEN
Swedish Chapter
Hans Lindstrom, 46-31-166794

• SWITZERLAND
Swiss Chapter
Contact Max Hugelshofer
ERNI & Co., Elektro-Industrie
Stationsstrasse
8306 Bruttisellen
01/833-3333

SPECIAL GROUPS

• **Apple Corps Forth Users Chapter**
1st & 3rd Tues., 7:30 p.m.
1515 Sloat Boulevard, #2
San Francisco, CA
Dudley Ackerman
(415) 626-6295

• **Baton Rouge Atari Chapter**
Chris Zielewski (504) 292-1910

• **FIGGRAPH**
Howard Pearlmutter
(408) 425-8700

• **NC4000 Users Group**
John Carpenter (415) 960-1256

ANNOUNCING

FORTH INTEREST GROUP
ROUNDTABLE

NOW AVAILABLE

THROUGH

GE INFORMATION SERVICES USA

GEnie™

General Electric Network for Information Exchange

* OVER 400 DOWNLOADABLE FILES OF FORTH INFORMATION & CODE
* ON-LINE REAL TIME CONFERENCING
* E-MAIL CONTACT WITH OTHER FIG MEMBERS

SPECIAL SIGN-UP FOR FIG MEMBERS ONLY

$18.00

Includes GEnie Manual plus 3 FREE Hours on GEnie

Using Your Modem Call 1-800-638-8369, Type "HHH" (CR)
Following the U# prompt, type "XJM11849,GENIE" (CR)

See Page 17 For Details

Forth Interest Group
P.O.Box 8231
San Jose, CA 95155