VOLUME X, NUMBER 2

JULY/AUGUST 1988

FORTH DIMENSIONS

OBJECT-ORIENTED FORTH

DESIGNING DATA STRUCTURES

STEP-TRACING fig-FORTH

LINEAR AUTOMATA

FORTH
DIMENSIONS

EDITORIAL

Charles Keane sent me a note on GEnie, to this effect:

"At it's last meeting, the ANS Forth Technical Committee (X3J14) voted to constitute itself as a Speakers Bureau for FIG Chapters, specifically on the subject of current standardization activity. It also designated y'r ob't servant as the clearinghouse for this effort. Interested chapters may contact me on GEnie (address C.KEANE), by phone (518-274-4774), or U.S. Mail (515 – 4th Avenue, Watervliet, NY 12189-3703)."

Invite a scapegoat to dinner, anyone? Seriously though, folks, this sounds like a great way to get a good, close look into the horse's mouth (so to speak). I suspect that any speaker from X3J14 could relate enough about the ANS process and technical tradeoffs to enliven and enlighten any meeting.

* * *

We have been looking for material related to object-oriented programming. There's a world full of people who think we'd be using objects, if we had any class. Mike Elola has kindly offered to bring objects to light in several articles. His first appears herein, and explains the fundamental concepts and terminology of object-oriented programming.

Like Forth, objects can be tough to appreciate without enough hands-on tinkering to provide, at the least, a gestaltic moment or two. To that end, Rick Hoselton provides F83 code that supports object-oriented programming. (Other versions are also welcome, and Mike Elola will be developing one with that series of articles.) He offers another view of the general subject, leaving it to the reader to develop some illustrations of the real usefulness of objects. We continue to welcome well-chosen examples and stories about object-oriented programming in Forth.

Any Forth programmer remotely interested in this topic must read Dick Pountain's book, *Object-Oriented Forth*. It's mandatory — even the introduction is good. Add it to your library even if you're just generally interested in Forth techniques, especially ones involving data structures (Academic Press, 1987).

* * *

I hope you can attend this year's Forth National Convention. Until this year, it always has been held in the vicinity of San Francisco. This was natural — most of its early organizers lived in that area, and the strong FIG chapters there supported it vigorously as volunteer staff and as attendees, speakers, and exhibitors. Not incidentally, some local FIG members also attended the business group and board meetings at which the convention was planned.

For years, there have been brief discussions about moving this keystone event of FIG's year to another locale. That would give local Forth programmers and vendors a chance to use the event as a showcase of their Forth-related work, and would provide the local technical community with a chance to learn about contemporary Forth products and practices. Besides, the inevitable infusion of techno-gossip and code-riddled repartee would give the local Forth community an infusion of ideas and a sense of perspective. But such discussions were usually short-lived, coming too late in the planning year and without local leadership or an actual plan.

Martin Tracy lives in southern California and is a member of the Board of Direc-

(Continued on page 38.)

Forth Dimensions
Published by the
Forth Interest Group
Volume X, Number 2
July/August 1988
Editor
Marlin Ouverson
Advertising Manager
Kent Safford
Design and Production
Berglund Graphics

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* (ISSN 0884-0822) is published bimonthly for $24/36 per year by the Forth Interest Group, 1330 S. Bascom Ave., Suite D, San Jose, CA 95128. Second-class postage paid at San Jose, CA. POSTMASTER: Send address changes to *Forth Dimensions*, P.O. Box 8231, San Jose, CA 95155."

LETTERS

ANS Process Offers Fairness

Dear Marlin:

This is in response to the letter from Gary Chanson (*FD* X/1) regarding an American National Standards Institute (ANSI) standard for Forth and the process which gets us there.

I am NASA's representative on the ANSI Accredited Standards Committee (ASC) X3J14. I have attended every hour of every meeting of X3J14. I was as apprehensive and suspicious before the first meeting as anybody had a right to be. My motivation was and is simple: I am a Forth user with important applications in view, and feel I have a stake in the outcome of the standards process.

Gary's letter might have left some readers with the impression that this process has been commandeered by the big Forth vendors for their own purposes. Or that users have been left out. I would like to offer my testimony to the contrary.

By my own observation I can attest that X3J14 was formed in accordance with ANSI rules. It does, in fact, have an ANSI charter to draft a proposed ANSI standard. As far as I can tell, it is operating in scrupulous regard for the letter and the spirit of ANSI rules.

Now, regarding ANSI rules, these are well-honed and rather inflexible. Space does not permit a full run-down on them here, but they are basically concerned with full public scrutiny; with guarantees that all input from any source be considered; and with ensuring that adequate time intervals are allowed for public response to proposed standards. These rules have resulted from decades of experience in developing standards. They are designed to balance the numerous interests that are always involved, whether the field is nuts and bolts or computer languages. In my judgment, these rules prevent chaos: by adhering to them, a standard can come into being; and a standard, by definition, reduces chaos. Further, the rules seem to be about the only realistic approach to achieve fairness.

Thus, even if they wanted to, or attempted to, the big Forth vendors probably would not have found it possible to commandeer the ANSI process.

The other misimpression that might have been left by Gary's letter is really just the flip side of the above concern — that users have no voice. Actually, the representation on X3J14 puts users in a near majority position. At my last count, there were nine producer members and 11 consumer members, with two others designated as "general interest." Membership, by the way (as has been stated widely and often), is completely open to anyone willing to pay the membership fee ($175), and who is willing to work, put in the time, and travel to the meetings.

Certainly, none of us likes the idea of a new standard drafted in secret by a self-appointed clique and then handed down as though from on high. Gary's letter represents the opinions of the Boston [chapter of the] Forth Interest Group, to the effect that they are ticked off by the past and don't want it repeated. My message is simple: take advantage of the new rules and the new process; get involved; make photocopies of the technical proposal and comment forms published in *Forth Dimensions* X/1 and in *Dr. Dobb's Journal* #137 (April 1988) and submit proposals and comments; seriously consider the possibility of becoming a member of X3J14 to represent the points of view shared by you and your group. And be prepared in a year or so to get a copy of the draft proposed ANS Forth, study it, and comment on it formally. That, too, is part of the ANSI process. Your comments must receive due consideration and must be answered formally for the process to continue.

Perhaps the reaction expressed in Gary's letter stems from the way our existing standard, Forth-83, was brought forth (or handed down?). That process, of course, was not the ANSI process. The rules were quite different. And with hindsight we can see they were not adequate to prevent dissatisfaction. I understand all this, but that was five years ago, and five years is an eon in the world of computers. Grudges someday must be laid aside and realities be consulted. And we do have new realities staring us in the face (e.g., 32-bit microprocessors which were not real five years ago), and more realities to face shortly (e.g., optical storage).

I believe that achieving ANS Forth will be an important event. It surely is inevitable. X3J14 is working very hard to make it a high quality achievement, one that will indeed have the broad support of users such as myself. But once again, your contributions are more than welcomed. They are expected!

James L. Rash
NASA
Goddard Space Flight Center
Greenbelt, Maryland

(Shattuck's screens.)

```
Scr # 37        KERNEL.BLK
\ Task Dependant USER Variables                        03Apr88cws
USER DEFINITIONS
VARIABLE  TOS          ( TOP OF STACK )
VARIABLE  ENTRY        ( ENTRY POINT, CONTAINS MACHINE CODE )
VARIABLE  MPAGE        ( MEMORY PAGE )
VARIABLE  JUMP         ( ADDRESS OF RESTART OR NEXT TASK )
VARIABLE  LINK         ( LINK TO NEXT TASK )
VARIABLE  SP0          ( INITIAL PARAMETER STACK )
VARIABLE  RP0          ( INITIAL RETURN STACK )
VARIABLE  DP           ( DICTIONARY POINTER )
VARIABLE  #OUT         ( NUMBER OF CHARACTERS EMITTED )
VARIABLE  #LINE        ( THE NUMBER OF LINES SENT SO FAR )
VARIABLE  OFFSET       ( RELATIVE TO ABSOLUTE DISK BLOCK 0 )
```

```
Scr # 22        CPU68000.BLK
\ Multitasking low level                               03Apr88cws
LABEL (PAUSE)   (S -- )
   IP SP -) MOVE  RP SP -) MOVE   ( push ip, rp )
   UP bank L#) D7 MOVE  D7 A0 LMOVE   ( load up )
   SP A0 ) MOVE  ( sp to tos )  8 A0 LONG ADDQ WORD
   A0 ) D7 MOVE  D7 A0 LMOVE   ( point to next task)
   A0 ) JMP  C;   ( jump to next task )
LABEL RESTART   (S -- )
   SP )+ A0 LMOVE   ( pop return address, current link )
   8 A0 LONG SUBQ WORD  A0 UP bank L#) MOVE   ( get up )
   A0 ) D7 MOVE  D7 SP LMOVE   ( restore stack )
   SP )+ D7 MOVE  D7 RP LMOVE   ( restore rp )
   SP )+ D7 MOVE  D7 IP LMOVE   ( restore ip )
   NEXT  C;
ENTRY LINK !   ( I point to myself )
```

```
Scr # 23        CPU68000.BLK
\ Manipulate Tasks                                     04Apr88cws
HEX
4EF9 CONSTANT JMPL#   \ op word for a long jump
4EB9 CONSTANT JSRL#   \ op word for a long jump to subroutine
DECIMAL
: LOCAL   (S base addr -- addr' )   UP @ -   +   ;
: @LINK   (S -- addr )   LINK @  ;
: !LINK   (S addr -- )   LINK !  ;
: SLEEP   (S addr -- )   DUP LINK LOCAL @  OVER JUMP LOCAL !
                         JMPL# SWAP ENTRY LOCAL !  ;
: WAKE    (S addr -- )   RESTART OVER JUMP LOCAL !
                         JSRL# SWAP ENTRY LOCAL !  ;
: STOP    (S -- )   UP @ SLEEP   PAUSE   ;
: SINGLE   (S -- )   ['] PAUSE >BODY ['] PAUSE !   ;
: MULTI   (S -- )   UP @ WAKE   (PAUSE) ['] PAUSE !  ;
```

(McBrien's screens.)

```
Screen 210
( DISFORTHer primitives. Retyped by Chris McBrien  20 Sept 1987.
  Adapted from Hewlett Packard's 9835 FORTH User's Manual.

  DISFORTH will decompile a Forth word into it's component
  words or tell you if it is a USER, VARIABLE or CODE
  definition. To ease typing, DISFORTH is renamed SEE   )

1 VARIABLE STRINGLIST    ] (.") [
2 VARIABLE TERMINATORS   ] ;S (;CODE) [
4 VARIABLE BRANCHES      ]     (LOOP) (+LOOP) BRANCH 0BRANCH [
5 VARIABLE LITERALIST    ] LIT (LOOP) (+LOOP) BRANCH 0BRANCH [

: ELEMENT?   ( n \ list ELEMENT? pos)  ( list is searched for n)
  DUP 2+ SWAP @ 2 * OVER + SWAP
  DO I @ OVER = IF DROP I 0 LEAVE THEN 2 +LOOP
  IF 0 THEN ;                                                  -->
ok

Screen 211
( DISFORTHer...                                     Page 2 of 3 )
: PRINT-WORD   ( pfaddr PRINT-WORD   next addr )
  CR DUP U.                                ( address )
  DUP @ DUP U. SPACE                       ( cfa )
  SPACE DUP 2+ NFA ID.                     ( name )
  DUP STRINGLIST ELEMENT? IF               ( if inline string )
      SWAP 2+ COUNT 2DUP TYPE              ( then type it out )
      + 2 - SWAP THEN
  DUP ' COMPILE CFA = IF SWAP 2+ DUP @ 2+ NFA ID. SWAP THEN
  DUP LITERALIST ELEMENT? IF
      SWAP 2+ DUP @ ROT BRANCHES  ELEMENT? IF OVER + THEN U.
      ELSE DROP THEN 2+ ;

: PRINT-DEF   ( PFA  PRINT-DEF )
  BEGIN DUP @ TERMINATORS ELEMENT? 0= WHILE
        PRINT-WORD REPEAT PRINT-WORD DROP ;                    -->
ok
```

No TRAPs in His Multitasker...

Dear Marlin,

I want to thank you very much for publishing the article by Robert J. Eager, "Relocatable F83 for the 68000" (*FD* IX/6). I know that some people don't want to see such machine-specific articles in *Forth Dimensions*, but this one really helped me. I have a copy of F83 modified for the Atari ST by George Morison. Mr. Morison did a wonderful job of porting F83 to the Atari, with the same basic idea used by Mr. Eager. Unfortunately, both the single-step debugger and the multitasker caused the system to bomb, so I did without them. With the help of Mr. Eager's article, I was able to fix the debugger almost immediately, and used it to tackle the multitasker.

After hours of constant bombing, I decided there must be something about 68000 traps that I just didn't understand. It occurred to me that I could add another couple of bytes to the user area to allow the use of the `JSR` instruction rather than the `TRAP` instruction. This means a little more complexity for the words `WAKE` and `SLEEP`, but the code works, is easy to understand, and avoids some extra stack popping required by the trap instruction, so it may even run faster. `WAKE` now puts a `JSRL` instruction into `ENTRY` and the address of `RESTART` into a new user variable called `JUMP`. `SLEEP` puts a `JMPL` instruction into `ENTRY` and the address of the next task (taken from `LINK`) into `JUMP`. The included code is specific to the 68000 but I imagine the same idea would work with any processor, but without requiring any knowledge of traps and exception vectors.

Let's continue to hear more about multitasking in *Forth Dimensions*, and how about some articles about implementing multi-user Forth as well?

Sincerely,
Charley Shattuck
1509 Gerry Way
Roseville, CA 95661

Visible Forth (with no exceptions)

Dear Editor,

With reference to Rich Franzen's "The Visible Forth" (*FD* IX/3), the `EXCEPTIONS` in screen 17 do seem to make the application rather non-portable, at least until the user has sorted out the addresses of the exception words.

Although I claim no originality for the application submitted, I have cleaned it up and gotten rid of one major typing error.

```
Screen 212
( DISFORTHer...                                       Page 3 of 3 )
: DISFORTH      ( DISFORTH cccc             eg: DISFORTH VLIST )
  CLS                                       ( Clear the screen )
  CR [COMPILE] ' DUP NFA ID.                 ( get PFA of cccc )
  DUP NFA C@ 64 AND IF ."  ...is an IMMEDIATE word" THEN
  DUP CFA @ [ ' . CFA @ ] LITERAL = IF      ( colon definition )
      PRINT-DEF
  ELSE DUP CFA @ [ ' FENCE CFA @ ] LITERAL = IF
       ." ...is a USER variable.  OFFSET = " @ . CR
  ELSE DUP CFA @ [ ' 0 CFA @ ] LITERAL = IF
       ." ...is a CONSTANT.  VALUE = " @ . CR
  ELSE DUP CFA @ [ ' USER CFA @ ] LITERAL = IF
       ." ...is a VARIABLE.  CONTENTS = " @ . CR
  ELSE ." ...is a CODE definition. " CR
       DROP THEN THEN THEN THEN CR ;
: SEE   BASE @ >R HEX  DISFORTH R> BASE ! ;        ( SEE  VLIST )
ok
```

```
Screen 231
 ( TRIAL... To test the resolving of a BRANCH                    )

: TRIAL   10 0 DO   CR ." BRANCH TEST"
                 LOOP CR ;
ok
```

```
TRIAL
58DB 4D5   LIT A
58DF 8CB   0
58E1 584   (DO)
58E3 2025   CR
58E5 A68   (.") BRANCH TEST
58F3 553   (LOOP) B1D8
58F7 2025   CR
58F9 596   ;S
 ok
```

```
58D0 05 85 54 52 49 41 CC B7 58 0A 07 D5 04 0A 00 CB ..TRIAL7X..U...K
58E0 08 84 05 25 20 68 0A 0B 42 52 41 4E 43 48 20 54 ...% h..BRANCH T
58F0 45 53 54 53 05 E3 58 25 20 96 05 04 44 55 4D 50 ESTS.cX% ...DUMP
5900 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
5910 20 20 20 20 20 20 20 20 20 20 20 20 20 0B 59 25              .Y%
 ok
```

(McBrien's screens, continued.)

```
Screen 200
  1 ( ELEMENT?   Decompile high level Forth definitions          )
  2
  3 1 VARIABLE STRINGLIST    ] (.") [
  4 2 VARIABLE TERMINATORS   ] ;S (;CODE) [
  5 0 VARIABLE BRANCHES      ]                                   [
  6 5 VARIABLE LITERALIST    ] LIT (LOOP) (+LOOP) BRANCH OBRANCH [
  7
  8 ( The cfa of a word along with it's relevant list 'ELEMENT?'
  9   gives the address of the cfa in that list  )
 10 : ELEMENT? ( cfa \ list ____ addr in the list )
 11          ( 0593 BRANCHES  element? 71EE  ...in my system )
 12          DUP 2+ SWAP @ 2 * OVER + SWAP
 13          DO I @ OVER = IF DROP I 0 LEAVE THEN 2 +LOOP
 14          IF 0 THEN ;
 15
 16                                                          -->
 ok


Screen 201
  1 ( PRINT-WORD    More decompiling words                        )
  2 : PRINT-WORD   ( pfa ____ next pfa )
  3   CR DUP U.                                  ( print the pfa )
  4   DUP @ DUP U. SPACE                         ( print the cfa )
  5   SPACE DUP 2+ NFA ID.                       ( print the name )
  6   DUP STRINGLIST ELEMENT?                    ( if an inline string )
  7       IF SWAP 2+ COUNT 2DUP TYPE + 2 - SWAP      ( type it out )
  8       THEN DUP ' COMPILE CFA =
  9            IF SWAP 2+ DUP @ 2+ NFA ID. SWAP
 10            THEN DUP LITERALIST ELEMENT?
 11                 IF SWAP 2+ DUP @ ROT BRANCHES ELEMENT?
 12                    IF OVER +
 13                    THEN U.
 14                 ELSE DROP
 15                 THEN 2+ ;
 16                                                            -->
 ok
```

Listing One is the original, semi-working version but, as can be seen, the `BRANCH` resolving is not correct according to the test word `TRIAL`. (Joke `SEA TRIAL`s.) I am a novice Forth programmer and have been unable to cure this branching problem apart from the vicious hack in Listing Two, in which I removed the `WORDS` from `VARIABLE BRANCHES` simply because the branch address seemed to be double what it should be; so I removed the duplicate branch words, and it worked. The test word `AA` in Listing Two is resolved correctly. If anyone can throw some light on this slight problem, I — for one — would learn a little more. Also, I feel this version would be more portable than Rich's: it originally came from a Hewlett-Packard 9835 application later modified for an HP 86, which uses an octal-based processor.

My system is basically a Forth-79 kernel, with additional words for an MS-DOS system running on a Hewlett-Packard 150. (Notice that screens' line numbers go from one through 16, not zero through 15. This does mean that `.LINE` is one off when used. Why don't people stick to a standard?

Chris McBrien
1. Milton of Straloch
Newmachar,
Aberdeen, Scotland

Errata and Improvements to a 6502 Assembler

Dear Marlin,

While using the assembler I described in *Forth Dimensions* (IX/5), I have discovered several bugs. The first of these was due to my ignorance of some opcode procedures; the others were just errors.

There are a number of operations, such as LDA, for which the lists of available addressing modes include:

Absolute, X
Absolute, Y
Zero page, X

but not Zero page, Y.

I hadn't realized that the missing Zero page, Y addressing mode could be invoked by using an absolute address reference to zero page (e.g., 00E1). In trying to modify the assembler to automatically compile an absolute zero-page address when appropriate, I found a few more bugs. So I decided to simplify the logic sequence and correct the screens. Screens 2, 3, 5, 8, and 9 have small changes; screens 4 and 6 have massive changes based on use of the new words `?LEGAL`, `?ZP`, and `?IMM`.

The suggested improvement is the use of equates. One of the conveniences of a conventional assembler is the provision for using names for addresses; e.g.:

```
COUT EQU $FDED
TEMP EQU $E1
```

In the Forth assembler, `COUT` and `TEMP` can be defined as constants, but are needed only temporarily. If such constants are defined (either before or after the assembler is loaded) after space has been allotted for the assembler and before the dictionary pointer has been reset to the top of the core vocabulary, they will be available to the assembler but will be forgotten along with it when the vocabulary linkage is changed after assembly is complete.

Sincerely,
Chester H. Page
1707 Merrifields Drive
Silver Spring, Maryland

```
ASSEMBLER SCR # 1
\ Assembly sample                                          27JUN87CHP
\ Conventional format
\  LDA #0
\  LDY #$80
\ L1 STA 300,Y
\  DEY
\  BPL L1
\  JMP NEXT

\ Format for this assembler
\ ASSEMBLE TEST
\ 0 # LDA, 80 # LDY, 101 300 ,Y STA, DEY, 101 BPL, GONEXT
\ END

-->


ASSEMBLER SCR # 2
\                                                          08MAR88CHP
HEX
VOCABULARY ASSEMBLER
ASSEMBLER DEFINITIONS
VARIABLE MODE
VARIABLE MODE.KEY
\ The allowable numbers of labels and references is controlled
\ in the rest of this screen
14 ARRAY LABEL.TABLE              \ Provide for 20 labels, and
CREATE REF.TABLE 0 , 0 , 56 ALLOT    \ for 30 references
VARIABLE REF.POINTER
: CLEAR.TABLES 15 1 DO 0 I LABEL.TABLE ! LOOP
    REF.TABLE 3 + REF.POINTER ! ;
VARIABLE LONG.ADDR
-->


ASSEMBLER SCR # 3
\  Modes                                                   09MAR88CHP
: ZP 0 MODE ! 0 MODE.KEY ! ;      \ Adds 4 to opcode
      \ ZP is default mode
: ,X 1 MODE ! 1 MODE.KEY ! ;      \ Adds 14  (zero page,X)
: ,Y 2 MODE ! 202 MODE.KEY ! ;    \ Adds 14 - LDX, STX,  only
: X) 3 MODE ! 4 MODE.KEY ! ;      \ Adds 0     (ZP,X)
: )Y 4 MODE ! 8 MODE.KEY ! ;      \ Adds 10    (ZP),Y
: #  5 MODE ! 110 MODE.KEY ! ;    \ Adds 8     Immediate
: ,A 6 MODE ! 20 MODE.KEY ! ;     \ Adds 8     Accumulator
: )  7 MODE ! 40 MODE.KEY ! ;     \ Adds 2C - Indirect JMPs only
\    8                              Adds C  - Absolute address
\    9                              Adds 1C - Absolute,X
\    A                              Adds 18 - Absolute,Y
CREATE ADD.TABLE      \ Indexed by mode value
  1404 , 0014 , 0810 , 2C08 , 1C0C , 18 C,
-->


ASSEMBLER SCR # 4
\ A is a given address                                     09MAR88CHP
\ C is address returned by opcode mnemonic
: ?LEGAL ( C---C) DUP 1+ C@ MODE.KEY @ AND FF AND
 ABORT"  Illegal Opcode" DUP C@ 20 =          \ Check for ,A
   0= IF OVER 100 U< 0= IF MODE.KEY @ 0C AND
   ABORT"  Illegal Indirect" THEN THEN ;

: ABS.ADDR DUP 1+ @ MODE.KEY @ DUP 3C AND
ABORT" Illegal address" DUP 40 = IF DROP DROP ELSE AND 200 =
   IF -1 MODE +! THEN  8 MODE +! THEN 1 LONG.ADDR ! ;
: ?ZP ( C---C) MODE.KEY @ 20 = 0= IF OVER 100 U< 0=
 OVER 1+ C@ MODE.KEY @ OVER OR 262 = SWAP 3F = OR OR
  IF ABS.ADDR THEN THEN ;

: ?IMM DUP 1+ @ MODE.KEY @ AND 100 = IF -2 MODE +! THEN ; -->
\ Special treatment of immediate with CPX, CPY, STX, or STY,
```

```
ASSEMBLER SCR # 5
\                                                    09MAR88CHP
: LABEL.SAVE FF AND DUP LABEL.TABLE @         \ Not new label?
   ABORT" Duplicate label"
   HERE SWAP LABEL.TABLE ! ;            \ Save label address

: LC1 SP@ S0 4 - = IF SWAP LABEL.SAVE THEN ;
: LC2 SP@ S0 6 - = IF ROT  LABEL.SAVE THEN ;

: COMPILE.ADDRESS  ( A---)
    DUP FF00 AND 100 =    \  Is it a label?
    IF HERE REF.POINTER @ 0 OVER C! \ Full address label needed
      1+ !                          \ Save compilation address
      3 REF.POINTER +!    \ Advance for next entry
    THEN LONG.ADDR @
    IF , ELSE C, THEN ; \ Compile absolute address or ZP byte
  -->
```

```
ASSEMBLER SCR # 6
\ CREATE operators for defining mnemonics            09MAR88CHP
\ Multimode opcodes
: M/CPU CREATE 2 ALLOT C, , DOES> 0 LONG.ADDR ! LC2 ?LEGAL
   ?ZP ?IMM
     C@ MODE C@ ADD.TABLE + C@ + C,    \ Adjust opcode
  MODE.KEY @ 20 = 0= IF COMPILE.ADDRESS THEN ZP ;

\ Single-mode opcodes
: CPU CREATE 2 ALLOT C, DOES> LC1 C@ C, ZP ;

: BRANCHES CREATE 2 ALLOT C, DOES> LC2
  C@ C, C,
  HERE 1- REF.POINTER @ 1 OVER C! \ Branch offset needed
  1+ !                            \ Save compilation address
  3 REF.POINTER +! ZP ;           \ Advance for next entry
 -->
```

```
ASSEMBLER SCR # 7
\ Second pass replaces stored label targets          21JUL87CHP
: SECOND.PASS
    BEGIN -3 REF.POINTER +! REF.POINTER @ DUP 1+ @
          \ Find label compilation address
  DUP WHILE DUP C@ DUP LABEL.TABLE @  \ Label address
       3 ROLL C@       \ Word-or-byte flag
    IF 2 PICK - 1-          \ Offset
       DUP ABS  7F >
IF DROP CR ." Branch to " 100 + . ." is too far"
."  (or label is missing)" SP! QUIT
      THEN ROT C!
    ELSE ROT !
    THEN DROP REPEAT DROP DROP ;

 -->
```

```
ASSEMBLER SCR # 8
\  Definitions of mnemonics                          08MAR88CHP
0060 61 M/CPU ADC, 0060 21 M/CPU AND, 0060 C1 M/CPU CMP,
0060 41 M/CPU EOR, 0060 01 M/CPU ORA, 0060 E1 M/CPU SBC,
0060 81 M/CPU STA, 0060 A1 M/CPU LDA,
025E 02 M/CPU ASL, 025E 42 M/CPU LSR,
025E 22 M/CPU ROL, 025E 62 M/CPU ROR,
027E C2 M/CPU DEC, 027E E2 M/CPU INC,
016F E0 M/CPU CPX, 016F C0 M/CPU CPY,
036D A2 M/CPU LDX, 016E A0 M/CPU LDY, 027D 82 M/CPU STX,
007E 80 M/CPU STY, 007F 20 M/CPU BIT, 003F 40 M/CPU JMP,
00 CPU BRK, 18 CPU CLC, D8 CPU CLD, 58 CPU CLI, B8 CPU CLV,
CA CPU DEX, 88 CPU DEY, E8 CPU INX, C8 CPU INY, EA CPU NOP,
48 CPU PHA, 08 CPU PHP, 68 CPU PLA, 28 CPU PLP, 40 CPU RTI,
60 CPU RTS, 38 CPU SEC, F8 CPU SED, 78 CPU SEI, AA CPU TAX,
A8 CPU TAY, BA CPU TSX, 8A CPU TXA, 9A CPU TXS, 98 CPU TYA,
-->
```

(Letters screens continued on page 22.)

DESIGNING DATA STRUCTURES

MIKE ELOLA - SAN JOSE, CALIFORNIA

Forth includes all the fundamental tools needed to create data objects. With these tools, you can create innumerable different kinds of such objects.

For this series of articles, our focus will be on the data objects that are most portable across CPUs of different bit widths, and on the ability to share operations between related types of objects. Without these concerns for portability and pooling of operations, designing new data objects has been guided by two main criteria: the simplicity with which frequently associated operations can be implemented (which also affects the speediness of such operations) and the memory compactness of the layout (which often inversely affects the simplicity of the associated operations).

New designs should be evaluated with respect to all these criteria. In this discussion, the performance and compactness of the code will only be mentioned when new design approaches threaten to compromise them too much.

The many topics about data objects include: how much data typing is supported by Forth; what constitues a Forth data type; how portability issues converge with data typing issues; and how data typing can be implemented. Another topic which is often treated too lightly is the choice of action (specified following `DOES>`) in user-supplied, data-declaration routines. This action is adopted by all data objects created with the parent declarator. Throughout this text, I will refer to this behavior as the "default" or "initial" operation.

The Quest for Reliable Object Designs

To be able to talk about goals such as "reliability," several basic terms must be understood precisely.

Objects are binary representations of numbers, dates, letters, or other abstractions. The individual bits that comprise the object are usually grouped into larger units, which can represent more than a Boolean on/off state. These bit-groupings help to structure the object.

"An object is simply a collection of properties."

The design of objects encompasses more than structure alone. Each structural component of an object is invested with a particular interpretation, which gives rise to the properties exhibited by the object. An object can be thought of as a collection of structural components and their associated interpretations. A more portable, or implementation-independent, way to view an object is simply as a collection of properties. For example, a signed integer has a sign property. The sign property arises from a particular component of the object, such as the interpretation of the most significant bit.

Identifying each context in which the object is intended to be used will help to determine the properties of the object. For example, assume you have to store phone numbers. While a sign bit would not serve any useful purpose in this context, a "work" or "home" discriminator might be useful. A 32-bit signed integer object could still be used for storing the numbers, but the object would not have a sign property. Instead, the bit normally associated with a sign could be interpreted as a work/home flag, an altogether different property.

Operations act upon an object by taking advantage of known properties of the object. For example, a multiply operation uses the sign bits of its operands to determine the sign bit of the result. This way, the properties of the result are consistent with the properties of the input objects.

If we know the properties of the resultant object, we may say that the object is reliable with respect to the operation. When the result is an object with unknown properties, the operation is unreliable with respect to the object. For example, a string concatenation operation is reliable when it properly accounts for the maximum-length property of the string into which the result is stored. By designing operations which respect the invariant properties of objects, we make our data objects "reliable." (See *Reliable Data Structures in C* by Thomas Plum for a more detailed discussion of this subject.)

The process of object design requires careful judgments about all the properties and operations an object should support. These properties and operations cannot be considered separately. The necessity for specific operations determines the choice of properties for an object. Likewise, the choice of properties impacts the operations that can be reliably performed upon an object. A string storage operator cannot reliably store a string unless the string variable includes a "maximum length" property, so that it can at least report error conditions arising from space limitations. So the design of objects alternates between consideration of the operations to be supported and consideration of the structural components that reliably support those properties.

Forth Data Types

Forth is a typed language, in terms of having many objects that share the same properties. Accordingly, operators and objects must be correctly paired. For example, the `EMIT` operation is only useful when applied to the correct object. The Forth programmer must oversee the proper matchup between operations and objects. Other languages also expect you to make the proper matchups, but they can provide a warning when you have made a mistake. Forth provides no such warning, unless you add the necessary code to make this possible. But Forth's lack of built-in type checking does not imply an absence of data types or any special dislike of data types.

The topic of Forth data types is often a sensitive one. Most authorities would say that the absence of strongly enforced data typing is bad. What they are really saying is that most programmers cannot keep track as well as the computer can of what they declare, and this is inarguably true. But by overcompensating for human frailty with strictly enforced type checking, languages become too confining.

Few languages besides Forth will let you make the final decision about whether an operator and an operand will be suitable for one another. So sets of operations (`COUNT`, `-TRAILING`, and `TYPE` for example) can be intermixed in ways that support a variety of objects, and with much greater efficiency in Forth than in most other languages.

Properties of Forth Data Objects

The properties of an object arise out of the unambiguous and stable interpretations we associate with each bit and byte of an object. Properties also include other facts about an object, such as a length (including component parts) and the layout. The layout properties of a multi-part object involve the order of the parts and their offsets from the start of the object.

These shared properties of data objects give rise to data types like integer variables, ASCII character codes, arrays, etc. As a designer, you need little more than consistency among a group of objects to establish data types. (Because it encapsulates a group of properties, even one instance of an object establishes a data type.)

"Variable" is a convenient label for a group of similar objects. The word variable is used to identify a type of object without redescribing it. The term helps to displace phrases like "16-bit, signed integer value." ("Variable"refers to the variable data type — the group of objects with the properties we commonly associate with Forth variables. `VARIABLE` refers to the Forth routine that creates instances of variables.)

In Forth, named data objects have a parent code field address (CFA) associated with them. Although the CFA (and associated `DOES>` phrase) is more a behavioral inheritance of the data object, that behavior is tightly bound to the object. As such, this behavior can be loosely considered a property of the object. If you think of the CFA as a physical subcomponent of the object, this idea gains more respectability.

The behavior of a constant is to return the value with which it was declared. That action can be considered an operation, rather than a property. Here's why: to design the initial operation for a constant, first you must take into account the width of the stored datum, so that the correct fetch operator can be engaged. Therefore, the width of the object is the fundamental property.

A data object need not have any operational property, as in the case of user variables. Memory has been allocated for user variables without a nearby CFA and label. Other examples of objects without built-in operational properties are disk buffers and headerless tables.

Reusability of Operations

Provisions for data typing block the compilation of an incorrect type of operator for an object, or else report a fatal error at run time. (Some compilers will perform type conversions automatically to avoid this error, but that digresses...) In object-oriented languages, provisions for objects assist in the selection of the correct type of operation through hierarchical data typing; if the current operation type is not found, an appropriate parent type operation may be selected. This operator-selection mechanism is called inheritance.

Careful design of Forth data objects also allows operations to be reused by different objects. The reusability of operations has received recent attention due to its introduction in object-oriented versions of established languages. In Forth, however, this kind of inheritance mechanism is overkill. When objects inherit operations, they are "enabled" for use. Without data abstraction or data type enforcement, Forth operations are always enabled for use with any object; but the selection mechanism is that creature known as the programmer.

Objects which have identical properties can be directly manipulated by a common set of operations. Objects may also be designed that share certain properties and not others. In such cases, you can often use a subset of the operations for both types of objects — particularly, the operations which engage only the property or properties shared by the different objects.

For example, an array of characters can be one object printed by `TYPE`, and a counted-string is another object that can be printed by `TYPE` when preceded by `COUNT`. You could also say that `TYPE` only works on one object (an array of characters), and it is incidental that the object may be part of another object. Whatever view you take, an array of characters is the property shared by both these objects, and at least that much can be clearly stated.

For this discussion, I will not consider parts of objects to be distinct objects. Rather, I encourage the reader to think of such a subcomponent as a distinct property. Any such properties can be shared by one or more different objects. In other words, treat the properties of an object as traits that must be individually accounted for by the applicable operations. Because `TYPE` only addresses the property of an array of characters, it can be applied to a variety of objects with that component property. (A side-effect of making operations property oriented rather than object oriented is that strict data type enforcement becomes more difficult.)

By designing objects to share important properties, we will also be able to design reusable operations for those objects. So a design strategy for data objects and their supporting operations may be:

> When designing data objects that are closely related to one another, choose layouts that are as regular (standard) as possible, which results in shared layout properties. Similarly, choose the initial operations so that their functionality dovetails with pre-existing operations. (Efficiencies are more likely to be realized when objects have as many shared

properties as possible.)

Such a strategy should yield a robust subset of general operations, as well as a minimal subset of object-specific operations.

Categorizing Operations

There are a couple of ways to categorize operations: as memory based or stack based. Since strings cannot be placed on the stack, string operations are memory based (although parameters for such string operations may be passed on the stack).

Later, we will categorize operations as object sensitive (or object specific) and object insensitive. An object-insensitive operation could be applied to many distinct, but related objects. `TYPE` was shown to be an object-insensitive operation which can be used across different string objects.

Tiers of Operations

A user variable is structured differently than a variable or a constant. The initial operation of a user variable fetches an address that points to the associated value. The net effect is the same as with a variable (yielding the address of a cell). However, a different initial operation was required, in order to account for the different layout property of a user variable: a pointer resides where the value would normally be. Once the address is placed on the stack, the fetch and store operators (`@` and `!`) can be used with either type of variable. So those operators are object-insensitive, because they can be used with several types of objects.

To be precise, these different kinds objects ultimately make reference to a cell and, therefore, exhibit cell properties. The cell is one of three tiers (or supertypes) of basic objects in Forth. Other tiers of operations are based upon the double and the character (or byte).

After the value associated with a cell or character value has been fetched onto the stack, other of the cell-oriented tier of operations can be applied, such as add, subtract, logical AND, logical OR, etc. Ultimately, many different objects are manipulated by the same set of Forth operators. In this way, Forth derives increased efficiency and compactness.

The ease with which different object layouts can be homogenized for use with a

(Continued on page 38.)

OBJECT-ORIENTED FORTH

RICK HOSELTON - HOUSTON, TEXAS

Once upon a time, while the sorceror was away, his apprentice magically made a broom carry water from a well into the house. When the job was done, the apprentice didn't know how to stop the magic broom, so he chopped it into a thousand pieces. But then, each piece began to carry water. The house was flooded, and the apprentice almost drowned. The sorceror himself had to straighten out the mess.

To write a complicated program, you definitely need to "chop it into pieces." But, as the sorceror's apprentice discovered, chopping up a problem just any old way sometimes makes it worse! You need a technique for breaking programs into *manageable* pieces.

How can you structure a program so that it is as simple as possible? Well, every useful program mimics some activity or event. An inventory program may simulate a warehouse operation; a game program might simulate an airplane flight or a poker hand. A program can't be simpler than the event it simulates. When your program's structure precisely matches the structure of the event it mimics, you have avoided useless complexity.

Events can be naturally divided into objects. For instance, an airplane flight is made of objects such as a plane, a pilot, and an airport. These objects act in ways determined by their natures. Airports stay in one place, planes must take off before they can land. Objects interact. When a pilot manipulates a plane's controls, he "sends a message" to the plane to bank or to climb. Complicated objects can be made of simpler objects. A plane can be considered to be made of an engine, control surfaces, etc.

Matching a program's structure to an event's objects is called object-oriented programming. Some supporters of object-oriented programming believe that programs should treat everything, even each location in memory, as an object. In practice, "object orientation" is a matter of degree. Object programming is a style or philosophy, as much as a formula.

Any computer language can be used to produce object-oriented programs, but some make it easier than others. Some programming languages, like Smalltalk, actually require an object approach. Forth is not a likely choice to rigidly enforce a programming discipline, but it can easily be extended to provide object programming support.

"Object programming is a style or philosophy, as much as a formula."

Objectives

Following Forth's minimalist philosophy, the routine should be brief. It should meet the common goals of object programming, and allow programmers to extend and customize it for their own use. The routine must not interfere with the current capabilities of Forth — the goal is to enhance the powers of Forth, not to bury them.

The routine must define *objects* to handle their own data with their own routines (*methods*). An object may exchange information with other objects by sending and receiving *messages*. It shouldn't directly access or change other objects' data. An object "obeys" or "acts on" messages by executing corresponding methods.

The routine must define methods for the objects. A method is a routine an object uses to manipulate its data. Executing a method is the way an object responds to a particular message. The same message may be used in different ways by different objects. For example, two objects named `GOLFBALL` and `TRUCK` might have different methods for the message `DRIVE`. The phrase `DRIVE GOLFBALL` would cause a completely different action than the phrase `DRIVE TRUCK`.

The routine should support *late binding*. Early binding means the system needs to know which object is to receive a message at the time it compiles the message call. With late binding, the application can wait until run time to decide which object should receive a message. The phrase

```
ENTERMETHOD DECEMBER?
IF ANNUALOBJECT
ELSE MONTHLYOBJECT THEN
```

would not work in an early binding system.

The routine should support *inheritance*. Sometimes, a group of objects can respond to the same group of messages with the same methods. And sometimes, a group of objects must share some data with each other. It's convenient to describe such groups of objects as *classes*. Common methods and common data can be described just once for the entire class, instead of once for each object. This is called inheritance.

Object Forth must execute quickly. If it is not fast, it is just not useful.

(Text continued on page 34.)

```
    0
\ OBJECT PROGRAMMING SUPPORT TOOLS     30 September, 1987



************************************************************
  The following routine is placed in the public
domain.  I give my permission for it to be used for
any legitimate purpose, free of charge.
  I make no warranty of any kind for this routine, and
bear no responsibility whatever for its use.

                     Rick F. Hoselton

************************************************************
```

```
    14
\ Data Structures:

OFFSET    #BYTES    METHOD format
0         2         next older brother METHOD pointer
2         2         MESSAGE number
4         n         METHOD's code


OFFSET    #BYTES   OBJECT format
0         2        father OBJECT address
2         2        youngest son OBJECT address + 4
4         2        next older brother OBJECT address + 4
6         2        youngest METHOD address
8         n        optional local data
```

```
    1
\ Loading

CREATE APPLICATION

     2  9  THRU     \ Object tools
    10 13  THRU     \ Testing - Demonstration


\S

This is a routine to assist FORTH programmers who
want to produce "OBJECT ORIENTED" code.  That phrase
seems to mean different things to different people.
Here is what it means to me.
```

```
    15
\ Loading

a "FORGETable" definition

The wordset to make OBJECTs and METHODs work.
A demonstration.




This routine is written to work with LAXAN & PERRY's F83.
Other FORTH implementations will probably require some
changes.  Especially, check METHOD:  ACTION and ?CREATE.
```

```
    2
\ ACTION

CODE ACTION  ( obj msg -- )
  AX POP   W POP    6 # W ADD
  BEGIN  0 [W] W MOV   2 [W] AX CMP  0= UNTIL
  4 # W ADD    0 [W] JMP   END-CODE

\S

: ACTION  ( obj msg -- )        \ Just like the CODE
  SWAP 6 +                      \ locate METHOD pointer
  BEGIN @ 2DUP 2+ @ = UNTIL     \ search for equal MESSAGE
  4 + NIP EXECUTE ;             \ execute the METHOD
```

```
    16
\ ACTION

This word finds the MESSAGE on the given OBJECT's
METHOD-list and performs the corresponding METHOD





This high-level definition does the same thing.
It is provided for documentation, and for those who
systems that aren't 8086 family based.  The speed loss
does not seem to be critical.
```

```
     3
\ ACTION

VARIABLE 'MSG                  \ Current MESSAGE # location

VARIABLE 'OBJECT               \ Points to current OBJECT

: ACT ( pfa msg -- )           \ What MESSAGES do.
  2DUP  'MSG @ !  'OBJECT !   ACTION ;

: ME ( -- ?? )                 \ Current OBJECT
   'OBJECT @ ;
```

```
     17
\ ACTION

Points to the last METHOD in the list.  Placing the MESSAGE
   number into this location ensures that a match will be found!
Points to the current OBJECT

Setup the OBJECT and MSG pointers, then go perform the
   METHOD requested for this OBJECT.

Place current OBJECT's address onto the stack.


Place current OBJECT's  FATHER's address onto the stack.
```

```
     4
\ Object addressing


: )OBJECT ( rel-addr -- addr )  \ Locate in current OBJECT
   ME +  ;

: )SUPER  ( rel-addr -- addr )  \ Locate in OBJECT's parent
   ME @ + ;

: LINK, ( addr -- )             \ Link here to addressed head
   HERE OVER @ ,   SWAP ! ;     \ addr points here, here points
                                \ where addr pointed
```

```
     18
\ Object addressing


Convert an OBJECT offset into a memory address.


Convert an offset in the current OBJECT'S father into a
memory address.

Useful for building links, and we use many.
```

```
     5
\ MASTER

: OBJECT)  ( -- )
   'OBJECT LINK,             \ make current and build father
   0 ,                       \ start with no sons
   2 )SUPER   LINK,          \ link up with brothers
   6 )SUPER @   ,  ;         \ inherit METHODs

CREATE MASTER
   MASTER 'OBJECT !          \ make master the current
   OBJECT)                   \ master is his own father !
   2 )OBJECT 6 ERASE         \ clear master
```

```
     19
\ MASTER

For building OBJECTS.





MASTER is the top OBJECT in the system.  All OBJECTs,
even MASTER, are descendents of MASTER!
```

6

```
\ (METHOD:)

: (METHOD) ( -- msg )        \ Build a MESSAGE
   CREATE HERE DOES) ACT ;

: ?CREATE  ( -- msg )        \ Get MESSAGE number
   )IN @   BL WORD FIND
   IF  NIP  )BODY ELSE  DROP )IN ! (METHOD) THEN ;

: (METHOD:)  ( -- )
   ?CREATE                   \ Be sure MESSAGE exists
   6 )OBJECT  LINK, ,        \ Link this MESSAGE number
   [ ' : @ ] LITERAL ,       \ Enter colon definition
   !CSP ] ;                  \ compile this METHOD
```

20

```
\ (METHOD:)

Build a MESSAGE header, and leave the PFA on the stack.
At run time, the MESSAGE will ACT.

If a MESSAGE has not been defined, define it.
Either way, leave the MESSAGE number (parameter field address)
on the stack.  BE CAREFUL not to use a name for a MESSAGE that
has already been used for anything but a message!

Set (or create) the MESSAGE number.
Compile this MESSAGE number and link up the METHOD chain.
Compile the code for this METHOD.
```

7

```
\ MASTER METHOD's

\ The system's base METHOD
(METHOD:) ANCHOR  ." I don't understand"  ;
' ANCHOR )BODY 2+  'MSG !

\ Building a METHOD is a METHOD
(METHOD:)   METHOD:  ( -- )
  (METHOD:) ;

\ Building a new OBJECT is a METHOD for the parent
MASTER METHOD: OBJECT: ( -- )
   CREATE    OBJECT) ;
```

21

```
\ MASTER METHODS

ANCHOR is always at the end of the METHOD chain.
Its MESSAGE number is set by ACT to the current MESSAGE.
So, if ACT finds no other matching METHOD, it uses this one.

This is the default METHOD for building METHOD's.
OBJECT's can have a different METHOD: if you define one.

This is the default METHOD for defining OBJECT's.
You may define a different OBJECT: to build
more complex types of OBJECT's.
```

8

```
\ .METHODS

: .METHOD ( link -- )
   CR  DUP  6 U.R   DUP @  6 U.R
   2+ @  DUP 6 U.R   2 SPACES BODY) )NAME .ID ;

MASTER METHOD: .METHODS ( -- )
  BASE @ HEX   6 )OBJECT
  BEGIN  @  ?DUP WHILE  DUP .METHOD REPEAT
  BASE ! ;
```

22

```
\ .METHODS

Display the name of a METHOD.

Displays all the METHOD's that have been defined for the
current OBJECT.
```

(Screens continued on page 33.)

CALL FOR PAPERS

for the tenth annual

FORML CONFERENCE

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

Following Thanksgiving, November 25–27, 1988

Asilomar Conference Center
Monterey Peninsula overlooking the Pacific Ocean
Pacific Grove, California U.S.A.

Theme: Forth and Artificial Intelligence

Artificial intelligence applications are currently showing great promise when developers focus on easy-to-use software that doesn't require specialized expensive computers. Forth's design allows programmers to modify the Forth language to support the unique needs of artificial intelligence. Papers are invited that address relevant issues such as:

Programming tools for AI
Multiusers and multitasking
Management of large memory spaces
Meeting customer needs with Forth AI programs
Windowing, menu driven or command line systems
Captive Forth systems—operating under an OS
Interfacing with other languages
Transportability of AI programs
Forth in hardware for AI
System security

Papers about other Forth topics are also welcome. Mail your abstract(s) of 100 words or less by September 1, 1988 to:

FORML
P. O. Box 8231
San Jose, CA 95155

Completed papers are due by October 15, 1988. For registration information call the Forth Interest Group business office at (408) 277-0668 or write to **FORML.**

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.

STEP-TRACING IN fig-FORTH

GENE THOMAS - LITTLE ROCK, ARKANSAS

User's of Laxen and Perry's F83 have a stepping utility invoked by the word DEBUG. It steps through definitions, displaying the stack contents at each step every time a key is pressed. Users of fig-FORTH and its derivatives have no such utility inherently resident in their systems.

During a meeting of the central Arkansas chapter of the Forth Interest Group, someone noted how nice it would be if fig-FORTH definitions could be stepped through; they suggested redefining ; (semi-colon). Those with a fair degree of Forth experience — and perhaps even novices, after a moment's reflection — will see that attempting to define

```
: ; KEY DROP ;
```

will result in a problem as soon as the first semi-colon is encountered by the compiler. Even if the above definition was renamed to [;] and additions made it a workable replacement for ;, a lot of editing work would be required to insert the new word when needed and to remove it when done. When I started this project, I made a rule: the finished application must not require any editing when it is used.

The Solution

The solution I finally arrived at uses : (colon) rather than ; (semi-colon), and vectored execution "tricks" the colon into being redefined. This has three advantages over the other methods I tried. First, no debugging word needs to be edited in and out. Second, stepping mode can be toggled on and off without recompiling. Third, the colon itself does not have to be recompiled.

Listing One is the step-trace application. Listing Two contains a few support words, which must be loaded before the code in the first listing if your system doesn't already have them.

Now let's examine the step trace code, beginning with screen three of Listing One. The word DEBUG is executed and the routine to be debugged is recompiled. Executing DEBUG replaces the value in the first PFA address (COL:ADR) of : (colon) with the CFA of [:] (STEP:VAL). The definition of [:] beginning on line 18 will now be used when a colon is encountered in definitions compiled after DEBUG execution. The remaining PFA addresses in the definition of : will not be executed because of the R> DROP on line 25.

> ***"[:] will now be used when : (colon) is encountered..."***

The stepping function will be taken care of when STEP? is called by [:] (STEP?'s CFA was pushed into [:] on line 23). When STEP? is called (see line 7), the variable DO-STEP? is checked to see if words are to be stepped; if so, it uses the top of the return stack to display the name of the word and displays the contents of both stacks. STEP? then stops and waits for a keypress (line 11). If the keypress is a B (or b), BREAK is executed; otherwise, the next word is stepped. (See *FD* V/1 for a full explanation of the BREAK/GO tool in screen two of Listing Two.)

Here is how : (colon) would look if decompiled before execution of DEBUG:

```
: :   ?EXEC !CSP
  CURRENT  CONTEXT
  CREATE   (;CODE)
    HERE 2- ! ]   ;
    IMMEDIATE
```

And, after executing DEBUG:

```
: :   [:]   !CSP
  CURRENT  CONTEXT !
  CREATE   (;CODE)
    HERE 2- ! ]   ;
    IMMEDIATE;
```

But, because of the construction of [:], the debug version of : acts as if it were defined like:

```
: :   [:]   ;   IMMEDIATE
```

Thus, through the magic of vectored execution, we are able to toggle between two alternate versions of : (colon), compiling under whichever we choose.

The default state of STEP? is off. After compiling a routine for use with DEBUG, the word STEP is executed to toggle to the stepping mode. STEP may be called as often as desired. Whenever you are unsure whether compiling is set to normal or to debug, invoke ?STEP to find out. Of course, the normal compiling condition of the colon is restored by RESTORE:COL.

A display of the return stack contents is of little value unless there is an easy way to identify the word to which those numbers (PFA return addresses) belong. While in the BREAK state, or at any time when not executing, the word NAME on screen four of Listing One will provide the needed information. Feeding any valid address from a parameter field to NAME will produce the name of the word to which that parameter field belongs. Sometimes the return stack contains items like DO LOOP indices. Giving NAME an invalid PFA has never crashed my system, but a memory check location in addition to DEF-END could be added to stop NAME when the bottom of the dictionary is reached. It is also well to remember that a return stack number may be equivalent to a PFA address and yet not actually be one. A DO LOOP index, for example, may be equal to some PFA address. Actually, NAME will respond correctly when given any address from a word's dictionary entry, except the last PFA address containing the CFA of ;S (EXIT). In that case, NAME will produce the name of the following word in the dictionary.

Four words in the step-trace application are intended to be executed from the keyboard: STEP, STEP?, RESTORE:COL, and DEBUG. The words NAME, BREAK, GO, .S, and .RP are independent of the step trace in the same way as words like R> and DROP.

When you are debugging the step-trace application itself, avoid crashes by executing RESTORE:COL before forgetting and recompiling. After the step trace is up and running, crashes will not occur if you forget to RESTORE:COL and recompile the routine you are debugging.

Compatibility

The definition of NAME assumes that each dictionary entry's LFA is followed by its NFA. If the NFA comes first in your system, change the 4 + on line 57 to 2+.

The definition of .RP assumes that the return stack grows downward in memory; an adjustment will be necessary for systems in which that stack grows upward.

I believe the definition of [:] will work in most systems, even if the : colon is defined differently than in fig-FORTH. If not, you will need to decompile your colon. Using that decompilation, insert the code on lines 21 – 25 at the appropriate place.

Gene Thomas edits the Comment Line, the newsletter of the Central Arkansas FIG Chapter, and is a registered polysomnographic technologist at the Sleep Disorders Center at the University of Arkansas for Medical Sciences.

Listing One.

```
Beginning scr #43
 0. \ Scr #1: Step trace                                  Gene Thomas Feb86
 1.   0 VARIABLE  DO-STEP?
 2. : STEP   \ -- | user; toggle step mode
 3.     SPACE  ." Step is "   DO-STEP?  DUP @  \   Toggle & display
 4.     IF  OFF   ." off."                     \   If on, turn off
 5.       ELSE  ON   ." on."                   \   If off, turn on
 6.     THEN  ;
 7. : STEP?  \ -- |vectored to from [:]
 8.     DO-STEP? @                      \   Step mode on?
 9.     IF  CR  R ( R@)                 \   If so copy pfa adr
10.      CFA NFA ID.                    \   Display name and stacks
11.      .S  .RP   KEY DUP  66 =        \    then stop and await key
12.      SWAP 98 =  OR                  \     press before continuing
13.      IF BREAK THEN                  \   If key = B or b then
14.     THEN  ;                         \    break, else continue
15. -->                                       stepping cycle

16. \ Scr #2: Step trace                                             gt Feb86
17.     ' : CFA @  CONSTANT CFA:
18. : [:]  \ -- |setup to vector colon for step execution
19.     ?EXEC !CSP CURRENT @ CONTEXT ! CREATE
20.       \ normal definition of colon to here
21.     CFA:                          \   Inserts your colon's cfa in
22.     HERE 2- !                     \    next adr
23.     ' STEP? CFA DUP @             \   Insert stepping instruction
24.     HERE 2- !  ,                  \    in the next adr
25.     R> DROP   ]  ;                \   Drop ret adr to original col
26. : remind  CR  ." Re-compile under current colon."
27.   0 VARIABLE STEP-MODE
28. : ?STEP   \ -- |user; show compiling condition: normal/debug
29.     CR  ." Compiling "   STEP-MODE @   IF  ." under step mode."
30.     ELSE  ." under normal mode."  THEN  ;
31. -->

32. \ Scr #3: Step trace                                            gt Feb86
33.   ' [:] CFA  CONSTANT STEP:VAL       \   Vectors used by debug
34.   ' :   @    CONSTANT COL:VAL        \    and restore:col
35.   ' :        CONSTANT COL:PFA        \   Contents toggled by
36.                                      \    debug and restore:col
37. : RESTORE:COL   \ -- |user; set for normal compiling
38.     remind    COL:VAL    COL:PFA !    STEP-MODE OFF ;
39. : DEBUG  \   -- |user; set for compileing  under step vector
40.     remind    STEP:VAL    COL:PFA !    STEP-MODE ON  ;
41. -->
42.   To debug: forget routine if not compiled under debug mode,
43. execute DEBUG and re-compile the routine.  Routine can now be
44. run under STEP (step on ok), or STEP (step off ok).  When
45. debugged: forget routine, execute RESTORE:COL and recompile.
46. Definitions are now compiled under normal conditions.  Step
47. Trace screens can remain in system.

Beginning scr #46
48. \ Scr #4; Step trace augment                                    GT may85
49.
50.   ' ;S CFA  CONSTANT DEF-END       \  ;S = EXIT in some systems
51.
52. : NAME  \ any-pfa-adr -- |user; show name of word to which pfa
53.                           \ belongs - is an independent word
54.     CR
55.         BEGIN
56.          2- DUP @
57.          DEF-END =                 \  Find end of prev dict entry
58.         UNTIL  4 +                 \   and jump forward to nfa
59.     ." Adr is in: "  ID.  ;        \  Show name of word pfa is in
60.
61.
62. ;S          TI-FORTH, an extension of FIG FORTH
63. END                                                         Step trace
```

(Thomas's screens, from previous page.)

```
Beginning scr #41
\ Scr #1; Non-destruct return stack display            GT may85
                Some def's you may need
: .RP  \ -- |user; display contents of return stack
    BASE @  HEX                         \  Save base
    CR  RP@ 2-  R0 @ 2-   ." R| "       \  Get limits of r-stack
    DO  I @ U.  -2  +LOOP               \  Display contents
    BASE !  ;                           \  Restore base

;S ------------------------------------------------------------
Alternate def for .RP
: .RP  BASE @  HEX  R0 @ 2+  RP@ DO  I @ U.  ( 2 +LOOP)
    LOOP  BASE !  ;
  (r-stack grows up in memory:                Gray Smith)

RP@ for TI-FORTH users (load scr #74, code):
HEX   CODE  RP@   0649 , C64E , 045F   DECIMAL
\ Scr #2: Step trace support; modified for Step Trace, gt
\ Break point tool from Forth Dimensions vol V no. 1
\ Debugging tools: BREAK & GO                  WF     13DEC81 KRH
\ by Frank Seuberling, 5/4/81
   0 VARIABLE CHECK
: BREAK  ( -- )   CR  RP@ 4 -  CHECK !      \  R-stack security
    0 BLK !                                 \  Take input from
    BEGIN                                   \   terminal
     QUERY  INTERPRET   ." aok"   CR
    AGAIN  ;
: GO  ( -- )   RP@   CHECK @  =             \  R-stack security
    IF  R> DROP  R> DROP                    \  Resume exec prog
     ELSE  ." Can't resume"  QUIT           \   unless rp@ has
    THEN  ;                                 \    changed

;S END                                        Step Trace Support
```

(Page's screens, from page 11.)

```
ASSEMBLER SCR # 9
\ More mnemonics and special definitions              09MAR88CHP
90 BRANCHES BCC, B0 BRANCHES BCS, F0 BRANCHES BEQ,
30 BRANCHES BMI, D0 BRANCHES BNE, 10 BRANCHES BPL,
50 BRANCHES BVC, 70 BRANCHES BVS,
: JSR, SP@ S0 4 - = IF SWAP LABEL.SAVE THEN DUP 20 C, ,
DUP 200 U< SWAP 100 U< 0= AND IF REF.POINTER @ DUP 0 SWAP C!
\ If 100 < addr < 200 then it is a label
  1+ HERE 2- SWAP !        \ Save compilation address
  3 REF.POINTER +! THEN ;

: ,, SP@ S0 4 - = IF SWAP LABEL.SAVE THEN , ;
: C,, SP@ S0 4 - = IF SWAP LABEL.SAVE THEN C, ;
: END SECOND.PASS CURRENT @ CONTEXT ! ?EXEC ?CSP ; IMMEDIATE
: GONEXT ['] NEXT >BODY JMP, ;
: ^ ' >BODY JSR, ;  -->  \ Useful in composite primitives
 \ e.g., ASSEMBLE PROGRAM ^ A ^ B ^ C GONEXT END

ASSEMBLER SCR # 10
\ Assembler concluded                                 19JUN87CHP

FORTH DEFINITIONS

: PRIM -2 ALLOT HERE 2+ , ;

: ASSEMBLE ?EXEC CREATE ASSEMBLER PRIM
   [ ASSEMBLER ] CLEAR.TABLES ZP !CSP ;

IMMEDIATE

DECIMAL
```

(Letters continued on page 32.)

fig-FORTH

LINEAR AUTOMATA

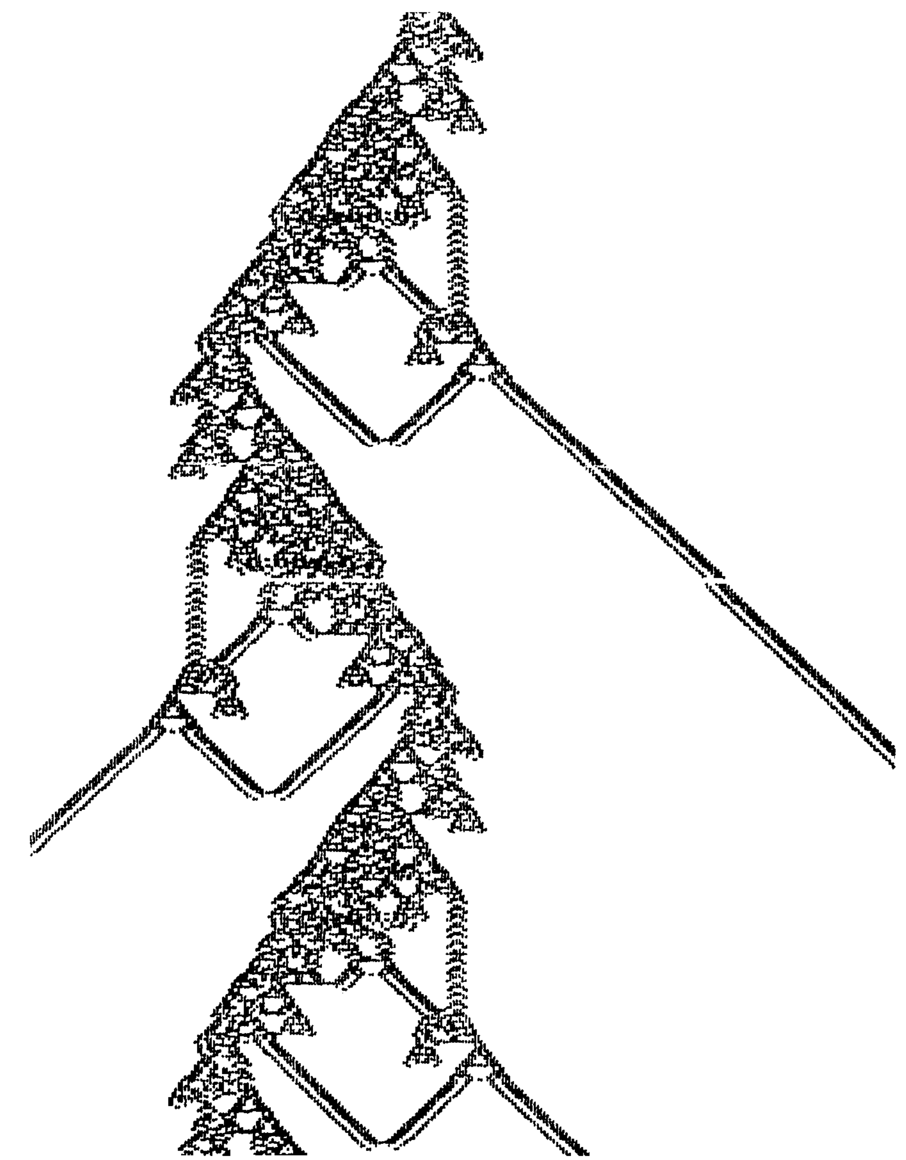
ANDREAS CARL - BERLIN, WEST GERMANY

The idea for the following program is from A.K. Dewdney in *Scientific American* (German edition, July 1985). He writes, "In a world of artificial computers, it is surprising to imagine that we might be surrounded by a variety of natural computers like water, wind, or wood. Such natural systems don't calculate in a conventional way, of course, but their structure makes arithmetic abilities a hidden possibility. Stephen Wolfram, physicist at the Institute for Advanced Study in Princeton, is as advocate of this thesis. He is of the opinion that a turbulent fluid or a growing plant are built of simple elements, but in the whole are so complicated that behavior cannot be reduced to mathematical terms. This irreducibility means they can store, transfer, and process information — they can calculate!"

"Cellular automata can calculate and simulate natural systems."

To demonstrate the arithmetic abilities of natural systems, he uses cellular automata. In looking for cellular automata which can both calculate and simulate natural systems, Wolfram confines himself to the simplest automata, those of one dimension.

These linear automata are defined by two constants and a set of rules, which define the transition from one generation of cells to the next. *k* gives the number of



states a cell can have; usually, it is two (to be or not to be). *r* is the radius; it determines the number of adjacent cells that will influence the subsequent state of a cell.

A table of rules gives the state of the next generation for every possible configuration. For example, for $k = 2$ and $r = 2$, there might be a rule which determines that a configuration like 0-1-0-1-1 leads the middle cell to become 1. For simplicity's sake, we can refrain from rules that Wolfram calls "total." Thus, a cell's next state depends only on the sum of the current states of all cells within radius *r*. In the example above, the sum can be between zero and 5, so a table of rules might look like:

Sum:	5	4	3	2	1	0
Next state:	0	1	0	1	0	0

If you read the next-state line as a binary number (e.g., 10100, which is $20 = 2^2 + 2^4$), you get the *code* of the rules. So this "linear automaton" is characterized by k=2, r=2, *code*=20. There are 64 different codes (tables of rules) for k=2 and r=2.

Now, if you apply this automat to the starting pattern 10111011, you will see that it moves like a signal to the right. In the world of this automat, it *is* a signal. Try to find other signals, patterns which produce or consume signals, and see what happens when two signals hit each other.

Try the automaton with k=2, r=3, and *code*=88 on the starting pattern 1111111111011. This is the famous cannon by J.K. Park: a signal-producing pattern which "shoots" to both directions every 119 generations. Good luck hunting for patterns!

Glossary

It should be fairly easy to convert the accompanying fig-FORTH screens into a dialect which will run on your computer. All you need is a Forth system with graphics capabilities.

`PLOT (x y --)`
Plots a point at the coordinates (x,y).

`?PLOT (x y -- f)`
Returns a flag, depending on whether a point is set or not.

`SUMME`
Calculates the sum of states for a given x coordinate.

`REIHE`
Compares `SUM` with `KODE` to decide whether to plot a point for any of the 320 x coordinates.

`AUTO`
Calculates the new generation for any of the 199 y coordinates.

`SET`
Sets a starting pattern into the first row (y=0) of the graphics display (bit map). (The address provided in the definition of this word is specific to the Commodore-64 on which it was written.)

```
#SCR 01

        ( LINEAR AUTOMAT )

   0 VARIABLE Y      0 VARIABLE SUM
   0 VARIABLE RADIUS 0 VARIABLE KODE

 : DUAL 2 BASE ! ;

 : SUMME RADIUS @ DUP 1+ SWAP -1 * DO 2DUP SWAP I + SWAP
                    ?PLOT 0= IF ELSE SUM @ 2 * SUM ! ENDIF
                    LOOP 2DROP ;

 : REIHE 320 0 DO I Y @ 1 SUM ! SUMME
                      SUM @ KODE @ AND
                      IF I Y @ 1+ PLOT ENDIF
              LOOP ;                             -->


#SCR 02

        ( CONT. )

 : AUTO 199 0 DO I Y ! REIHE LOOP ;

  : SET 256 /MOD 8352 C! 8360 C! DECIMAL ;
```

Real-Time Programming Convention

November 18 - 19, 1988
Grand Hotel, Anaheim, California

Call for Presentations

The 1988 Real-Time Programming Convention will be held at the Grand Hotel in Anaheim, California, and is sponsored by the Forth Interest Group.

The theme of this year's convention is *Real-time Programming Systems*. The invited speakers are Jef Raskin, head of the original Macintosh development team and inventor of the Canon Cat, and Ray Duncan, well-known author and expert on IBM PC Operating Systems. Both speakers have made extensive use of Forth, a language especially suited to real-time applications.

There is a call for presentations on topics in the following areas:

Programming Environments

Real-time Operating Systems
Language-oriented RISC machines
Parallel Processing
Languages for Data Acquisition and Analysis
Robotics and Real-time Device Control

Intelligent Devices

Intelligent Instrumentation
Working Neural Nets
Adaptive devices
Software Peripheral Controllers

Applications

Aerospace
Medical
Laboratory
Machine-vision
Digital Signal Processing
Robotics
Automation
Instrumentation

Presentations may be either talks or demonstrations. Talks are limited to fifteen minutes. Please submit an abstract of the talk and a request for any audio-visual assistance by October 15. Demonstrations may accompany the talk or appear separately throughout the convention. Please send a description of the demonstration and its requirements by October 15.

Abstracts and descriptions should be sent to: **Real-Time Programming Convention,** Forth Interest Group, PO Box 8231, San Jose, CA 95155.

Volume Eight Index

A comprehensive reference guide to all issues of *Forth Dimensions* published during the Volume VIII membership year. (Special thanks to indexer Mike Elola of San Jose, California.) See the FIG Order Form to order complete sets of back issues.

Advertisers Index

THE BEST OF GENIE

GARY SMITH - LITTLE ROCK, ARKANSAS

News *from the GEnie Forth RoundTable:* Beginning July 10, the Sunday on-line meetings at the "Figgy Bar" will feature a question-and-answer session for novices, with Leonard Morgenstern as chair for these tutorial conferences.

Since it is not unusual to see 10K or more of new messages on a given day, and this column is limited in size, you are only getting a peek at recent on-line activity. This time, the peek will be into the very lively standards category (Category 10).

Some still may not realize the X3/J14 Technical Committee has made the GEnie Forth RoundTable their home service. X3/J14 has the task of drafting a ANS standard Forth. Here, the very future of our language is being debated with a grand mix of knowledge, wisdom, and humor. This excerpt features a discussion centered around a proposal by Lee Brotzman. I hope it will encourage *you* to get involved.

Category 10, Topic 23, Message 76
Wed Mar 23, 1988 S.W.SQUIRES [scott]

Lee, I have some of the same suggestions that Leonard does for your file words. How about:

`OPEN` (addr - - file#)

File# could be a number or a handle or pointer or fcb or whatever would be in keeping with the specific computer/Forth system as long as it is consistent on that system. On a one-file limited system it would just leave the same number. Multiple files have been the norm for some time even in the simple Forth systems I've used. Typical case is reading in one file, manipulating it and writing it back out to another file.

`CLOSE` (file# - -)
`READ` (addr n1 file# - - n2)
`WRITE` (addr n1 file# - - n2)

`SEEK` and `FILEPOS` would require a file# as well. Would it be more beneficial to provide pointers with the `READ` and `WRITE` commands? I.e., `READ` (addr n1 file-offset file# - - n2) The more primitive the words, the more flexible they could be. Same thing with flags — would it just be more straightforward to leave a flag after every disk operation?

How about a create-file function? You'd probably need to provide a size parameter as well as an addr of the naming convention to allow for systems with unexpandable file sizes.

How about a request for the file size? This would allow a program to set aside the correct buffer size and to use the size for any calculations. —Scott

Category 10, Topic 23, Message 77
Thu Mar 24, 1988 L.BROTZMAN

Leonard and Scott,

Jerry Shifrin voiced the same concerns as yours when I uploaded my proposal to the East Coast Forth Board. I'll just reproduce my answer to him here:

===========================

Date: 03-23-88 (11:57) Number: 276
To: SYSOP Refer#: 273
From: LEE BROTZMAN Read: YES
Subj: HOST FILE ACCESS PROPOSAL
Status: PUBLIC MESSAGE

Yes, Jerry, I purposely avoided the subject of multiple files since I think that trying to pass file handles, of reference numbers or whatever, is so system specific that it becomes very difficult to standardize. This proposal is hard enough to get adopted as is; adding system-specific file handles would kill it for sure.

I don't agree that this proposal precludes multiple-file handling however, and let me explain why. I'll use Uniforth for my example, because that's what I know.

In Uniforth there is a user variable called `FCB`. `FCB` points to the file handle (file control block, reference buffer, whatever the OS in question uses) of the current open file. The value of `FCB` is changed by a set of words called: `CHANA`, `CHANB`, etc. To open two files simultaneously, for example, one would do the following:

```
CHANA OPEN file1.fth
CHANB OPEN file2.fth
```

A word that copies a line of text from one file to another would be something like this:

```
: COPY-LINE
  ( copy a line of text )
  ( from CHANA to CHANB )
 CHANA pad 80 RDLINE
  ( length - - )
 CHANB pad swap WRLINE drop ;
```

where I have used the Uniforth words `RDLINE` and `WRLINE` instead of my proposed words `READ` and `WRITE`. The code would be the same in either case.

If the proposal were changed to include file handles, I would anticipate changes like the following:

`OPEN` (- - fcb)
Open a file and return the file handle.

`CLOSE` (fcb - -)
Close the file pointed to by the file handle.

`READ` (fcb adr len1 - - len2)
As before except with file handle.

`WRITE` (fcb adr len1 - - len2)
As before except with file handle.

`SEEK`, `FILEPOS`, and `WREOF` would be changed similarly. Frankly, I don't see much difference in the ultimate use of these words. Returning the file handles means they must be saved somewhere in a variable. So the `COPY-LINE` above would become:

```
COPY-LINE
FCB1 @ pad 80 READ
FCB2 @ pad swap WRITE drop ;
```

(In fact the definition of `CHANA` is something like: `FCB1 @ FCB !` and `CHANB` is `FCB2 @ FCB !` for most, but not all operating system interfaces implemented.)

So you see, it isn't difficult to handle multiple files using the proposed word set. Perhaps I should say that in the proposal, in order to make clear what I already thought would be understood implicitly. I keep forgetting that other systems handle things in very different ways. Do you think I should also propose some standard means of file switching? It should be as generic as possible, because the manipulation of file control blocks is different for every operating system, while, in Uniforth at least, the ultimate top-level file operators like those above are uniform.

===========================

To continue, I would like to say that I prefer "file-switching" words like `CHANA` and `CHANB` to explicit references to file handles, because the explicit method is unnecessary and less self-documenting, and it follows the principle of "hiding data" àla Brodie's *Thinking Forth.*

Leonard, thanks for pointing out the deficiencies in language in my proposal. I see that it must be more carefully written to avoid misinterpretation. When I say `CLOSE` will "close the file currently open," I should say "...close the file on the current file I/O channel"—after I define what a file I/O channel is of course. :-)

The definition of `READ` should say that reading will stop "...when n1 bytes of data have been read, an end-of-file mark is encountered, or in the case of a variable..."

Finally, as I said above, my proposal isn't incompatible with "handles," it just assumed they are handled elsewhere (pun intended).

Scott, about file creation: much more than size and name go into file creation, like access method, logical record length, blocking factor, data type (binary, character, executable, etc.), protection, and on and on. That's a pretty big can of worms.

A request for file size is a good idea, and something I use a lot. I'll add it to the list. — Lee

Category 10, Topic 23, Message 78
Thu Mar 24, 1988 L.BROTZMAN
Greg,

Thanks for the tip on the proposal. I will try to amend the draft in light of the responses above and get it in the mail ASAP. While we're talking about proposals, I asked Martin Tracy whether discussion on my DO LOOP proposal could be postponed until the November TC meeting at Goddard Space Flight Center, since I plan to attend that meeting and would then be available to explain and answer questions. He said I should ask you, so I'm asking. (Actually, if there is a move afoot to go back to Forth-79 DO LOOPs, my proposal is obsolete, which is fine with me — I have no problems with the earlier DO structure).

Sorry about sounding irate re BLOCK in this topic. I really have nothing against BLOCK in host file operations, it has its place. I just don't think that it is a panecea.

My earlier postings about BLOCK in this topic have been (as far as I can recall without digging back into my log files) an effort to make it more compatible with the hosted environment, e.g. "undefined" block length, and releasing restrictions on buffer sizes. These are issues of little importance for standalone systems, but they could make life with BLOCK under an operating system a whole lot easier.

I don't think I ever said BLOCK wasn't suitable to access a database, just that it isn't the *only* suitable way. I expressed this explicitly in my last two messages, and I tried to be accommodating about saying that there are indeed times when BLOCK is the way to go — at least, that's what I wanted to say. (Damn electronic communications... bad E-mail, bad!)

Off the top of my head, the theoretical limit on throughput of a CD ROM drive is roughly 150 Kilobits/sec. I have not analyzed our system as to actual throughput (we have to make the disk first!), but if you have friends at JPL, the guy to ask there is Mike Martin of the Planetary Data Systems Group. He has produced two CD ROMs of astronomical images and character-table data, and has written software to support it on IBM PC/AT/XT clones under MS-DOS. He told me that his throughput on the PC rivals that of an unloaded VAX reading from a hard disk, but VMS is such a dog that I won't venture to interpret that statement.

The FITS files will be random access on the CD ROM. I would much prefer heavily indexed, flat text files but FITS has been foisted on me by NASA. Our first disk is simply a test of the CD ROM as storage and distribution medium, and FITS as a disk-based interchange format (currently, FITS is primarily for tapes, not disks, although several observatories have done some good work with disk-FITS already). The production schedule for this disk is too tight to allow more than minimal indexing for a few files (i.e., about 30 catalogs, totalling more than 50 files and 400 Mbytes; final selection isn't set until mid-May). Subsequent disks, assuming that funding is continued, will include index files into the FITS formatted data, and more sophisticated data-base software. By that time, I hope to have the Forth software advanced enough to stave off the higher-ups that think it should be in C.

You're right that the slow seek times are a real pain. Users are more than willing to put up with it, however, to get up to 600 Mbytes of direct-access storage on their PCs, all in one place at a relatively low cost. Drives are running about $700, and most CD ROM application disks are about $100-200 — ours will be distributed for cost of media only, of course — $40-50 at most. There are now a few vendors of drives that claim to cut the seek time by quite a bit, but I haven't seen the spec sheets yet. — Lee

P.S. Touché, JAX. A full-blown, Forth-based workstation environment couldn't end up any weirder or more esoteric than Unix, and *that's* pretty popular nowadays. Keep on trucking.

Category 10, Topic 23, Message 79
Thu Mar 24, 1988 S.W.SQUIRES [scott]
Lee,

I'd still prefer an explicit means of selecting a file. This would allow a variable (or better yet a TO-type variable) with a descriptive name for that particular program. (i.e. SOURCE, DESTINATION, ACCOUNTS, etc.) The potential problem with using the CHANA / CHANB is that the FCB is set until it is changed again. By looking at the source code for a program that did file access, you'd have to look back and determine what set it the last time, if you didn't do it in the actual word doing the file access. Likewise, debugging could be confusing if FCB was set by a stray word. By passing the FCB (or file#) explicitly, the program can actually become more readable. Also, the usage is up to the programmer and he can use arrays or other structures if he desires. —Scott

Category 10, Topic 23, Message 80
Fri Mar 25, 1988 J.SHIFRIN
Lee, I know I'll get confused trying to respond here and on the ECFB, but I still don't think your files proposal is very solid. Nothing against UniForth, but I think the CHANA/CHANB approach is both a kludge and a bit bizarre. Also, I believe it falls apart in a multitasking enviroment. I don't care what's passed as a file identifier, but I think it should be a single stack item — an address or i.d. number which uniquely refers to something (FCB, HCB, DCB, filename), implementation dependent, to describe the file being operated on

[Sorry about the awkward prose — I *hate* the GEnie editor and didn't want to get into it for cleanup. Should've composed this offline!]

Category 10, Topic 23, Message 81
Sat Mar 26, 1988 G.BAILEY1 [ATHENA]
Lee, your proposal (known as TP88-038) is in the pile for consideration at the May TC meeting, and I will state your request to postpone its consideration as a motion to commit it to the group that is working on control structure and looping issues. We will probably convene that group at least once in Rochester and it is probable that this group will not have concrete recommendations for some time. Unfortunately, it is difficult to indicate your willingness to

(McBrien's screens, from page 22.)

```
Screen 203
  ( DISFORTH  Decompile Forth words to their component words     )
  : DISFORTH      ( DISFORTH cccc )
    CR [COMPILE] ' DUP NFA ID.                      ( get PFA of cccc )
    DUP NFA C@ 64 AND                        ( check the precidence bit )
    IF ." ...is an IMMEDIATE word."
    THEN DUP CFA @ [ ' . CFA @ ] LITERAL =
         IF  PRINT-DEF                             ( colon definition )
         ELSE DUP CFA @ [ ' FENCE CFA @ ] LITERAL =
              IF ." ...is a USER variable.  OFFSET = " @ . CR
              ELSE DUP CFA @ [ ' 0 CFA @ ] LITERAL =
                   IF ." ...is a CONSTANT. VALUE= " @ . CR
                   ELSE DUP CFA @ [ ' USER CFA @ ] LITERAL =
                        IF ." ...is a VARIABLE. CONTENTS= " @ . CR
                        ELSE ." ...is a CODE definition " CR
                   DROP THEN THEN THEN THEN ;
  : SEE   DISFORTH ;
 ok
5540 54 37 04 96 05 82 41 C1 1D 55 0A 07 25 20 F5 04 T7....AA.U..% u.
5550 7B 55 68 0A 20 20 20 49 46 2E 2E 2E 20 6E 6F 6E {Uh.   IF... non
5560 20 7A 65 72 6F 20 70 72 69 6E 74 20 74 68 69 73  zero print this
5570 20 6C 69 6E 65 25 20 E4 04 9F 55 68 0A 1F 45 4C  line% d..Uh..EL
5580 53 45 2E 2E 2E 20 69 66 20 7A 65 72 6F 20 70 72 SE... if zero pr
5590 69 6E 74 20 74 68 69 73 20 6C 69 6E 65 25 20 68 int this line% h
55A0 0A 27 54 48 45 4E 2E 2E 2E 20 72 65 67 61 72 64 .'THEN... regard
55B0 6C 65 73 73 20 77 68 61 74 20 70 72 69 6E 74 20 less what print
55C0 74 68 69 73 20 6C 69 6E 65 25 20 96 05 88 44 49 this line% ...DI
 ok

AA
554C 2025   CR
554E 4F5   0BRANCH 557B
5552 A68   (.")   IF... non zero print this line
5575 2025   CR
5577 4E4   BRANCH 559F
557B A68   (.") ELSE... if zero print this line
559D 2025   CR
559F A68   (.") THEN... regardless what print this line
55C9 2025   CR
55CB 596   ;S  ok

 Screen 202
  ( PRINT-DEF     More DISFORTHer words                              )

  : PRINT-DEF   ( pfa --- )  ( word is decompiled from that pfa )
    BEGIN DUP @ TERMINATORS ELEMENT? 0= WHILE
          PRINT-WORD REPEAT PRINT-WORD DROP ;

  ( AA is a test word for SEE to check the branches are resolved
    corectly )
  : AA   ( n --- )
           CR
           IF ."   IF... non zero print this line" CR
          ELSE ." ELSE... if zero print this line" CR
          THEN ." THEN... regardless what print this line" CR  ;



 ok
```

(End of Letters screens.)

```
     9
\ .SONS

: .ME  (S n -- )
  CR SPACES   ME BODY> >NAME .ID ;

MASTER METHOD: (.SONS) (S n -- )
  DUP .ME   2+   2 >OBJECT
   BEGIN  @ DUP WHILE   2DUP 4 -   (.SONS)   REPEAT
  2DROP ;

MASTER METHOD: .SONS  ( -- )
   0    ME (.SONS) ;

MASTER METHOD: .ONE
  4 .ME ;
```

```
     23
\ .SONS

Display the name of the addressed OBJECT.


Display the name of the addressed OBJECT and, indented
the names of all his descendent OBJECTs.  This is a recursive
routine.  "LATE BINDING"  is very useful here.


Display the name of the current OBJECT and all his
descendents.

Display the name of the current OBJECT.
```

```
     10
\ Testing & demonstration

MASTER OBJECT: VEHICLE

VEHICLE METHOD: #WHEELS  8 >SUPER @ ;

VEHICLE OBJECT: BOAT      0 ,
VEHICLE OBJECT: CAR       4 ,
VEHICLE OBJECT: TRICYCLE  3 ,

CAR OBJECT: GREEN-MONSTER
BOAT OBJECT: QUEEN-MARY
\S
QUEEN-MARY #WHEELS  .
GREEN-MONSTER #WHEELS  .
```

```
     24
\ Testing & demonstration

A "superclass" of vehicle types

A METHOD for finding the number of wheels for a
  grandson of VEHICLE.
An OBJECT whose immediate descendants have no wheels
An OBJECT whose immediate descendants have 4 wheels
An OBJECT whose immediate descendents have 3 wheels

a famous car
a famous boat   ( Well, really it's a ship )

How many wheels does the QUEEN-MARY have ?
How many wheels does the GREEN-MONSTER have?
Never say CAR #WHEELS .  (#WHEELS isn't written for that.)
```

```
     11
\ Testing & demonstration

MASTER OBJECT: AUTOMOBILE



AUTOMOBILE METHOD: OBJECT:  (S n -- )
   CREATE OBJECT>      \ Build links
   0 ,                 \  8) Odometer mileage
   0 ,                 \ 10) Odometer mileage @ last fillup
   0 ,                 \ 12) gas in tank
     , ;               \ 14) miles-per gallon
```

```
     25
\ Testing & demonstration

A new example: AUTOMOBILE
Note that this AUTOMOBILE is not a son of VEHICLE.
We're on a new subject.

AUTOMOBILE type OBJECT's aren't quite the same as ordinary
OBJECT's.  They have some extra data appended.
To define an AUTOMOBILE OBJECT, the miles-per-gallon for
that OBJECT must be on the stack.
```

```
12
\ Testing & demonstration

AUTOMOBILE METHOD: DRIVE (S n -- )
   14 )OBJECT @   12 )OBJECT @  *    ( gas*mpg=range )  MIN
   DUP  8 )OBJECT +!                 ( increment odometer )
   DUP  14 )OBJECT @  /   NEGATE 12 )OBJECT +!
   ." I'm driving " . ."  miles " ;

AUTOMOBILE METHOD: TELL-GAS (S -- )
   12 )OBJECT @  CR
   ." I HAVE " . ." GALLONS IN MY TANK " ;

AUTOMOBILE METHOD: FILL-GAS (S n -- )
   12 )OBJECT +!   ME TELL-GAS  ;
```

```
26
\ Testing & demonstration

To drive our AUTOMOBILE n miles, we first check our range,
( gas times miles-per-gallon ) to see how far we can go.
Then we increment our odometer reading, decrease our fuel
and report just how far we drove.


Report the fuel in the tank.



Add fuel to the tank.
In a more complicated example, we might check the gas tank
capacity, reduce the driver's cash, etc.
```

```
13
\ Testing & demonstration

 7   AUTOMOBILE OBJECT: RACER
23   AUTOMOBILE OBJECT: SLOW-POKE

SLOW-POKE METHOD: TELL-GAS  CR ." It's a secret! " ;


\S

13 RACER FILL-GAS
100 RACER DRIVE
RACER TELL-GAS
SLOW-POKE TELL-GAS
```

```
27
\ Testing & demonstration

High performance, seven miles-per-gallon.
Low performance, twenty-three miles-per-gallon.

Some privacy for SLOW-POKE
This shows that different objects can use the same MESSAGE
name to produce different results.



Put some gas in the tank.
Drive for a while.
Report gas remaining.
```

Evaluation

I searched for an object-Forth support routine that met these objectives, but I didn't find any that really suited me. Neon provides some fine object tools, but it has changed so much that it isn't Forth any more. Vocabulary-based implementations of object Forth can be slow, and none that I examined support late binding. So, I wrote my own object Forth.

Rather than supporting an explicit class construction, this routine supports inheritance by causing each object to be the "son" of some other object. "Brother" and "cousin" objects can inherit data, data structure, and methods from the common "ancestors."

There don't seem to be any bugs left, but there is a "feature" I don't like: it is easy to misuse a method. A method might be designed only to be inherited. It might not work at all with the original object, but it is still possible to make that request. Another warning: don't use a name for a message if that name has already been used for something else. There is very little error checking; when you ask for a mistake, you usually get one.

Summary

Here is a fast, late binding, and free object-Forth support routine. It runs under Laxen and Perry's public-domain F83. Now you possess Forth, the world's most powerful programming language, and support for one of the world's most powerful conceptual tools: object-oriented programming.

Rick Hoselton is a professional Forth programmer with General Information Technologies, Inc. His work with Forth spans the last six of his seventeen years spent as a full-time computer professional.

FIG CHAPTERS

The FIG Chapters listed below are currently registered as active with regular meetings. If your Chapter listing is missing or incorrect, please contact Kent Safford at the the FIG office's Chapter Desk. This listing will be updated in each issue of *Forth Dimensions*. If you would like to begin a FIG Chapter in your area, write to the Chapter Desk for a "Chapter Kit and Application." **Forth Interest Group, P.O. Box 8231, San Jose, California 95155**

U.S.A.

• **ALABAMA**
Huntsville 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

• **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

North Bay Chapter
2nd Sat., 10 a.m. Forth, AI
12 Noon Tutorial, 1 p.m. Forth
South Berkeley Public Library
George Shaw (415) 276-5953

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) 454-1307

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
Bob Barr (408) 435-1616

Stockton Chapter
Doug Dillon (209) 931-2448

• **COLORADO**
Denver Chapter
1st Mon., 7 p.m.
Clifford King (303) 693-3413

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

• **FLORIDA**
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

• **INDIANA**
Fort Wayne Chapter
2nd Tues., 7 p.m.
I/P Univ. Campus, B71 Neff Hall
Blair MacDermid (219) 749-2042

• **IOWA**
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

• **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
Robert Washam
91 Weis Drive
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
Monroe Comm. College
Bldg. 7, Rm. 102
Frank Lanzafame (716) 462-3398

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

• OHIO
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 Ganger (513) 849-1483

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

• OREGON
Willamette Valley Chapter
4th Tues., 7 p.m.
Linn-Benton Comm. College
Pann McCuaig (503) 752-5113

• 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
Matt Lawrence
PO Box 180409
Austin, TX 78718

Houston Chapter
3rd Mon., 7:45 p.m.
Intro Class 6:30 p.m.
Univ. at St. Thomas
Russell Harris (713) 461-1618

• UTAH
North Orem Chapter
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.
Hal Clark (802) 453-4442

• VIRGINIA
First Forth of Hampton Roads
William Edmonds (804) 898-4099

Richmond Forth Group
2nd Wed., 7 p.m.
154 Business School
Univ. of Richmond
Donald A. Full (804) 739-3623

• WISCONSIN
Lake Superior Chapter
2nd Fri., 7:30 p.m.
1219 N. 21st St., Superior
Allen Anway (715) 394-4061

MAD Apple Chapter
Bill Horton
502 Atlas Ave.
Madison, WI 53714

INTERNATIONAL

• AUSTRALIA
Melbourne Chapter
1st Fri., 8 p.m.
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
Peter Tregeagle
10 Binda Rd., Yowie Bay 2228
02/524-7490

• BELGIUM
Belgium Chapter
4th Wed., 8 p.m.
Luk Van Loock
Lariksdreff 20
2120 Schoten
03/658-6343

Southern Belgium Chapter
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

Southern Ontario Chapter
Quarterly, 1st Sat., Mar., Jun., Sep., Dec., 2 p.m.
Genl. Sci. Bldg., RM 212
McMaster University
Dr. N. Solntseff (416) 525-9140 x3443

Toronto Chapter
John Clark Smith
PO Box 230, Station H
Toronto, ON M4C 5J2

Vancouver Chapter
Don Vanderweele (604) 941-4073

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

• FRANCE
French Language Chapter
Jean-Daniel Dodin
77 Rue du Cagire
31100 Toulouse
(16-61)44.03.06

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

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

• JAPAN
Japan Chapter
Toshi Inoue
Dept. of Mineral Dev. Eng.
University of Tokyo
7-3-1 Hongo, Bunkyo 113
812-2111 x7073

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

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

• SWEDEN
SweFIG
Per Alm
46/8-929631

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

SPECIAL GROUPS

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

(Data Structures, from page 14.)

tier of operations is a virtue of Forth. Charles Moore preserved a great deal of flexibility by manipulating values in steps: an object in memory is fetched to the stack, converted to a cell-length object (or multiples thereof), processed by whatever postfix operators are available to the object's supertype (you might say there are only two types at this point: doubles and cells), and then stored with the correct operators (back to three data types).

Conclusion

Other languages don't support the creation of new data types; they leave you with a small vocabulary of basic types and methods from which to build compound types such as records. The basic type vocabulary also tends to be a minimal one. Each of the few basic types available is usually quite different from the others. Contrast this with Forth, where you can create a dozen string data types if you need them. And within twelve different applications, you may find yourself needing them.

Forth doesn't presume to have discovered all the basic data objects or types needed to solve your problems. With its limited but extensible base, Forth provides the opportunity to create just the data type or data object you need.

Mike Elola is a published Forth programmer and a full-time writer at Apple Computer. Over the years, Mike feels, Forth has tricked him into believing that he is a computer scientist.

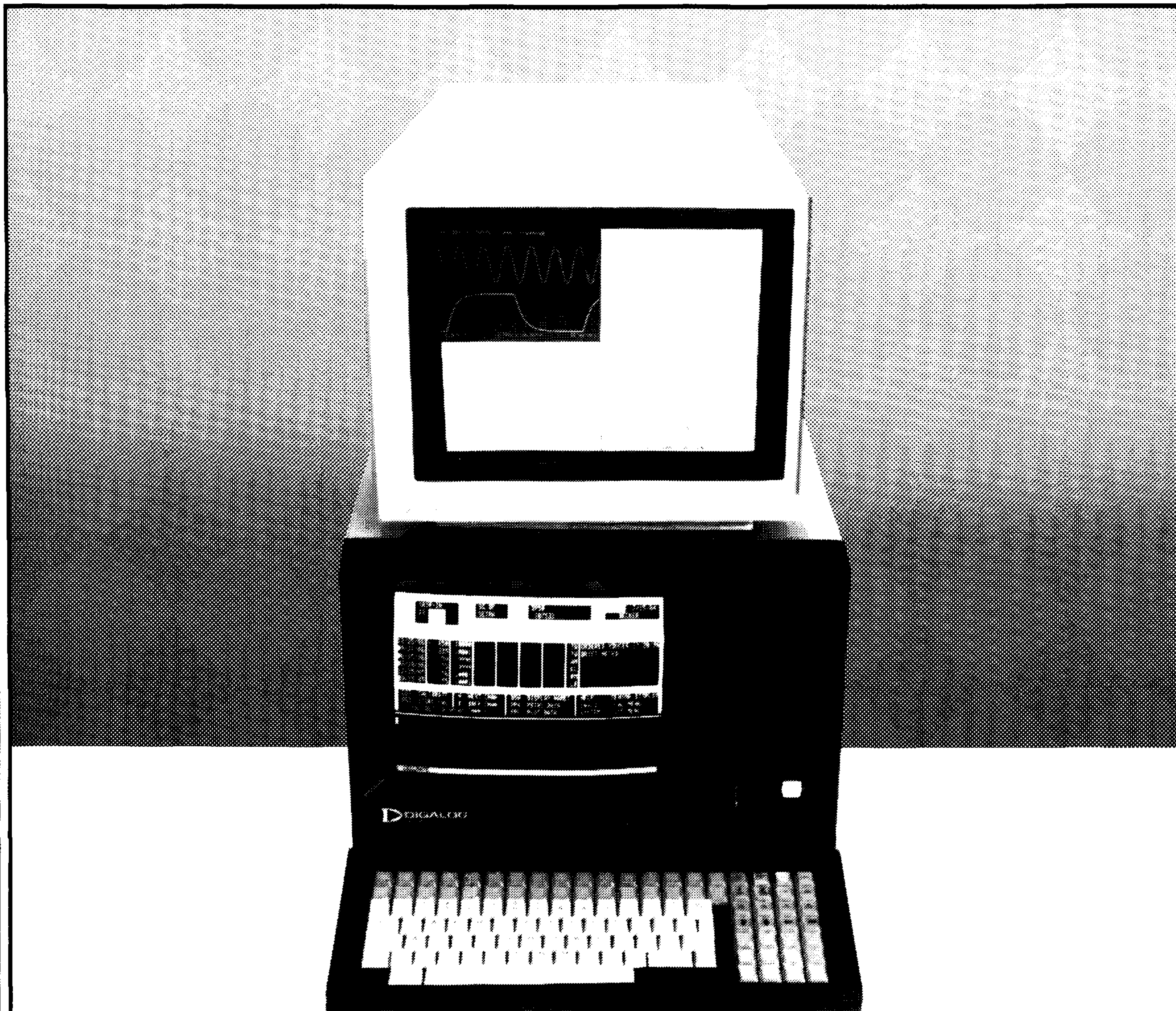
(Editorial, from page 3.)

tors of the Forth Interest Group. Under his leadership, the 1988 Forth National Convention ("The 1988 Real-Time Programming Convention," featuring banquet speaker Jef Raskin and keynote speaker Ray Duncan) will be held November 18 – 19 in Anaheim, California (across from Disneyland). Los Angeles and Orange County have local FIG members with the expertise, professionalism, and energy to lend to an exciting event. We are looking forward to traveling there, and hope to see you there. Bring along your computer and favorite language, too: the black-belt programmers contest offers a $1000 prize — write to FIG for a copy of the rules.

—*Marlin Ouverson*
Editor

(GEnie, from page 31.)

accept the Forth-79 definition in our audit trail, and unless *someone* generates a proposal to that effect there is no way it can even be considered. If you consider the Forth-79 loop behavior to be equally desirable, there is absolutely nothing wrong with submitting a separate proposal to that effect. There are plentiful cases where a submitter finds two mutually exclusive changes equally acceptable, and in such a case two proposals are easier to work with than would be a single proposal outlining two possibilities. Cheers —Greg.



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

Second Class
Postage Paid at
San Jose, CA