FORTH Dimensions

Volume 8, Number 3
September/October 1986
$4.00

Compiler Macros

Segmented Memory Models

Shuffled Random Numbers

Stack Numbers by Name

The Point Editor

Multi-Dimensions of Forth

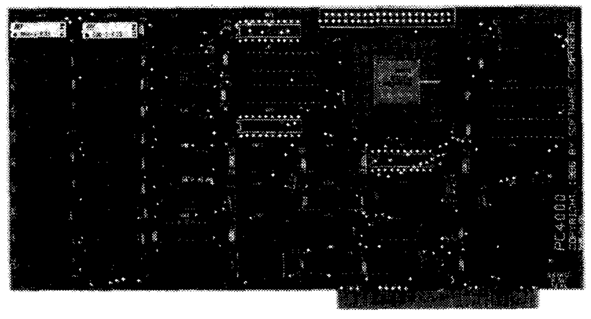
A PC DROP-IN BOARD WITH:

Novix Forth engine, 4 MIPS power, mini computer speed, parallel PC operation, 1/2 Mbyte on board, multi-tasking capability, software included. $1,495

YEAH, SURE.

It's finally here! The PC4000. Plugs into PC/XT or PC compatible. Comes with 4 Mhz clock. Upgrade to 5 Mhz by adding faster RAM and Clock. 16K of memory ported to PC bus for PC/PC4000 data transfer. Selectable 2K or 8K stacks or multiple stacks for multitasking.

Runs the Sieve in Forth in .09 seconds—2170 times faster than the Sieve runs on the PC in PC-Basic. Includes SCForth software package for software development. RAM on board can be used to extend host memory space. C software available.


PC4000

Contact:
Software Composers
Suite F
210 California Avenue
Palo Alto, CA 94306
415-322-8763

SOFTWARE COMPOSERS

Forth Dimensions
Published by the
Forth Interest Group
Volume VIII, Number 3
September/October 1986
Editor
Marlin Ouverson
Advertising Manager
Kent Safford
Production
Cynthia Lawson Berglund
Typesetting
LARC Computing

Forth Dimensions solicits editorial material, comments and letters. No responsibility is assumed for accuracy of material submitted. Unless noted otherwise, material published by the Forth Interest Group is in the public domain. Such material may be reproduced with credit given to the author and to the Forth Interest Group.

Subscription to *Forth Dimensions* is free with membership in the Forth Interest Group at $30 per year ($43 foreign air). For membership, change of address and to submit material for publication, the address is: Forth Interest Group, P.O. Box 8231, San Jose, California 95155.

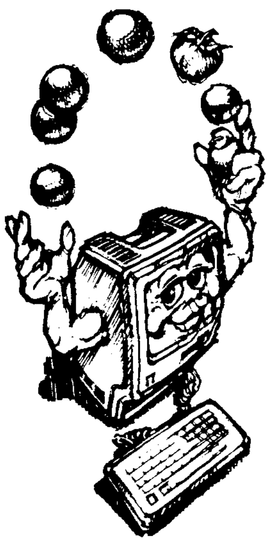
Symbol Table

	Simple; introductory tutorials and simple applications of Forth.
	Intermediate; articles and code for more complex applications, and tutorials on generally difficult topics.
	Advanced; requiring study and a thorough understanding of Forth.
83	Code and examples conform to Forth-83 standard.
79	Code and examples conform to Forth-79 standard.
fig	Code and examples conform to fig-FORTH.
FORML	Deals with new proposals and modifications to standard Forth systems.

FORTH Dimensions

FEATURES

DEPARTMENTS



Mach1™

Multi-tasking FORTH-83 Development Systems for 68000 and 68020-based microcomputers and industrial target boards.

Mach1 is a FAST, 32-bit subroutine-threaded implementation of FORTH. All Mach1 systems include:

- Unlimited multi-tasking--any number of background/terminal tasks are allowed.
- Local variables for readable, recursive, re-entrant programming.
- Standard text files--Any text-only editor/word processor may be used.
- A STANDARD Motorola 68000 assembler (infix) which supports forward label references.

Apple Macintosh™

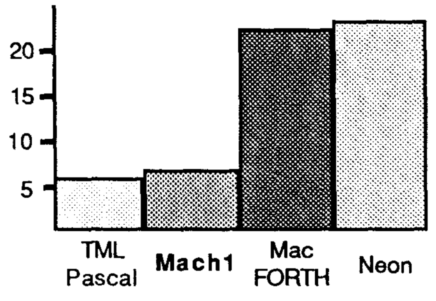
$49.95

$10 Demo Disk also available

Mach1 on the Mac features:

EASY creation of stand-alone applications
Complete toolbox support (including Mac Plus routines)
Macintalk speech driver support
Redirection of I/O to serial ports/devices
Graphics printing support
80-bit SANE floating point
68020 compatible

Comes with Switcher/Edit and 400 page manual



Sieve Times (seconds)

Atari ST™

$59.95

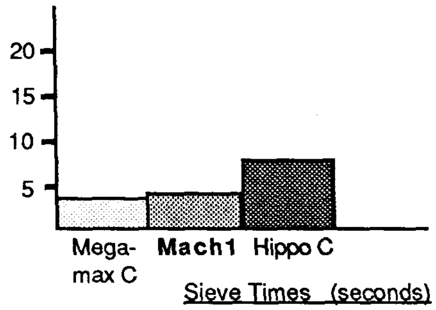
$10 Demo Disk also available

Mach1 on the ST features:

EASY creation of stand-alone applications
Full GEM and DOS support
Integrated GEM editor
68020 compatible

Comes with 300 page manual

Available July 15



Sieve Times (seconds)

Commodore Amiga™

($59.95 Available September 1986)

Industrial Target Systems

Mach1 in EPROM: (Available immediately on Gespac G-64 68000 board)

16K FORTH Kernel	$500
16K 68000 Assembler	$500
16K Dissassembler/Debugger	$500

Call for source licensing arrangements....

Executes sieve on 16MHz 68020 in 1.3 seconds

Palo Alto Shipping Company • P.O. Box 7430 • Menlo Park, California 94026

(800) 44FORTH - Sales • (415) 854-7994 - Support

VISA/MC Accepted • Include S/H on all orders ($5 US/Canada, $10 overseas) • CA res. add 6.5% tax

Letters to the Editor

What Forth Can Learn From C

A recent ad for an Amiga version of Multi-FORTH from Creative Solutions reminded me of Forth. I had almost forgotten about Forth because recent issues of *Computer Language* and *Dr. Dobb's Journal of Software Tools* treat Forth as if it no longer exists.

Wednesday night I went to a well-attended Amiga owners meeting to see a demonstration of a brilliant new piece of Amiga software from a company called B.E.S.T. (P.O. Box 852, McMinnville, Oregon 97128). The software will allow small businesses to do *all* their bookkeeping using an Amiga. B.E.S.T. began last fall using Lattice C and recently switched to Manx's Aztec C. When I asked these brilliant programmers if they had considered programming in Forth, they expressed the opinion that Forth is no match for C.

Are they right?

Besides, B.E.S.T. says they have developed their own auxiliary C tools that allow them to program on the Amiga twenty times faster than ordinary Aztec C.

There are lots of libraries of subroutines which can be purchased and incorporated into C programs to speed development time. Are there libraries of extra Forth "words" for sale to speed development? It seems to me that if Forth has been pushed aside by C, it may be because brilliant Forth programmers are spending their time reinventing the wheel (laboriously adding words to Forth and fiddling with how things are going on and coming off the stack) while C programmers are building upon the labors of those who have already invented the wheel, and they are making better progress.

Or, put another way, C is so difficult to learn and use that those who do use C are willing to pay for and include other programmers' work in their own programs. Forth, on the other hand, is so much easier to learn and use that Forth programmers have fallen behind while they happily work at adding their own words to extend their personal copy of the language.

I read in the August 1985 *Computer Language* that Philippe Kahn says Borland will offer Modula-2, BASIC, C and Ada languages, but not Forth. He won't offer Forth because, he says, "Forth is a religion ..."

Where can I read the definitive comparison of C and Forth? Are Forth programmers sharing words instead of encouraging Forth users to go off on their own to reinvent the wheel?

Awaiting your prompt reply,

Rich Kevin O'Brien
Renton, Washington

"Forth as a religion" was a way of poking fun at our own conviction, but like many analogies it has both detracted from real issues and outlived its usefulness. One real issue in need of more attention is Forth "subroutine" libraries. John James, in particular, has studied the factors involved in providing in individual modules the useful tools developers want and need. One assumes that a stable Forth nucleus will be critical to the creation of such off-the-shelf, plug-in extensions. Both system vendors and third parties may find a line of profit in this thinking, provided the work meets specific design criteria (see James' articles in Forth Dimensions *VII/4, VII/5 and VIII/2).*

By the way, Dr. Dobb's Journal *still publishes an annual Forth issue and, as of this writing, a periodical column about Forth. I haven't seen any "definitive comparison of C and Forth," but if one or more authoritative, objective authors could be found I'm sure the editors of* DDJ *and those of* Computer Language *would be interested in the idea. But I was talking with CL Publications' co-founder Craig LaGrow about this when he made the interesting point that the worst published comparisons are those that attempt to compare apples and oranges. Perhaps that is the area that will have to be addressed by anyone comparing Forth and C.*

—Editor

Teaching Forth: Testing TESTIT

Dear Marlin:

Mr. Apra's article, "Let's Keep It Simple" (VII/6) makes a good point, though perhaps not strongly enough. I have done some teaching and can testify to the real value of simplicity. Nevertheless, I would like to take exception to the efficacy of **TESTIT**, **IFTRUE**, **IFFALSE** and **ENDIT** as Mr. Apra proposes their use.

The points I would make are (refer to the Apra screens):

1. Students ought to learn how to recognize the need for **DUP** and **DROP**. It may be confusing that **TESTIT** will sometimes precede and at other times follow the test depending on whether the number is to be tested or the resulting flag needs to be **DUP**ed. In the definition of **EXAMPLE1** a **DROP** would need to be added if one of the two phrases were eliminated. It would be better to replace **TESTIT** with **DUP**. Then the correct action of eliminating **DUP** rather than adding **DROP** would be evident.

2. It is important that one's understanding of the **IFTRUE ELSE ENDIT** control structure be easily transferable to the **IF ELSE THEN** constructs. The major problem here, as I see it, is demonstrated by the definition of **EXAMPLE4**, which has the appearance of a **CASE** statement. I can imagine it might lead to further confusion. Of course, with a **CASE** statement, the ending **DROP** would be unnecessary, as well as the numerous **TESTIT**s (**DUP**s).

3. Mr. Apra correctly observes that "... the **IF ELSE THEN** syntax does not leave enough clues to where the parts of the conditional should go." But in the proposed syntax there is no clue to suggest when — and when not — to include **TESTIT**. Compare the definitions of **EXAMPLE1** and **EXAMPLE2**.

BRYTE FORTH

for the

INTEL 8031 MICRO-CONTROLLER

FEATURES

—FORTH-79 Standard Sub-Set
—Access to 8031 features
—Supports FORTH and machine code interrupt handlers
—System timekeeping maintains time and date with leap year correction
—Supports ROM-based self-starting applications

COST

130 page manual —**$ 30.00**
8K EPROM with manual—**$100.00**

Postage paid in North America.
Inquire for license or quantity pricing.

Bryte Computers, Inc.
P.O. Box 46, Augusta, ME 04330
(207) 547-3218

I offer the code on the enclosed two screens as the second step in the quest to simplify **IF ELSE THEN**, for the purpose of teaching, to an understanding of that conditional structure. I know from experience that such conditionals are difficult to grasp for many a beginner. Sometimes those of us who do understand forget that.

I believe the **IFTRUE OTHERWISE RESUME** concept offers a better chance to gain an understanding of the conditional's true structure. The examples I have included were designed to demonstrate how variable the structure of such conditionals can be. Replacing **DUP** with **TESTIT** in the examples will show why I think **TESTIT** is not so useful.

I offer my congratulations to Mr. Apra. I hope there are many like him searching for better ways to teach beginning young programmers. Finally, a word to the students: letters such as this one are not "put downs" of other authors. Progress doesn't stop with a first attempt at improvement; it is an evolution. I am confident that competent programmers will argue with both Mr. Apra and me. Some will even pooh-pooh it altogether. So be it. But, of course, most of them are not teachers.

Sincerely,

Gene Thomas
Little Rock, Arkansas

```
Listing 1
Screen #20
\ IFTRUE/IFFALSE, OTHERWISE (optional), RESUME          GT May86
 : IFTRUE  \ f -- (if tf execute following)
     [COMPILE] IF ;  IMMEDIATE
 : IFFALSE \ f -- (if ff execute following)
     COMPILE NOT [COMPILE] IF ;  IMMEDIATE
 : OTHERWISE  \ execute following if IFTRUE or IFFALSE not exec.
     [COMPILE] ELSE ;  IMMEDIATE
 : RESUME  \ -- (resume program here after IFTRUE-OTHERWISE)
     [COMPILE] THEN  ;  IMMEDIATE

 : ?EQUALSTEN  ( n -- )  DUP  10 =
     IFTRUE ." True =10" DROP  OTHERWISE ." False "  10 >
                                 CASE  0 OF ." <10"  ENDOF
                                       1 OF ." >10"  ENDOF
                                 ENDCASE
     RESUME ;

\ IFTRUE: scn #2; examples

 : ?EQUALSTEN  ( n -- )  DUP   10 =   DUP
     IFTRUE  ." Yes, "  SWAP   .   RESUME
     IFFALSE ." No, "    .   RESUME  ;


 : ?EQUALSTEN  ( n -- )  DUP 10 =
     IFTRUE ." True"  DROP        ( and proceeds to last resume)
        OTHERWISE  10 >           ( compare with n )
          IFTRUE ." Bigger than 10"
             OTHERWISE  ." Smaller than 10"
          RESUME
     RESUME  ;


```

Eighth Annual

Forth National Convention

November 21–22, 1986

The Doubletree Hotel at Santa Clara Trade & Convention Center
Great America Parkway and Tasman Drive, Santa Clara, California 95050

Conference Program

Forth Engines Research

Learn about the latest advances in Forth hardware and software from the creators and designers of the modern Forth engines.

Demonstrations • New Products • Tutorials • Chapter Activities

Forth Exhibits

Convention activities start Friday, November 21, at 12 noon.

Convention preregistration	**$15**	**Conference and Exhibit Hours**
Registration at the door	**$20**	**Friday Nov. 21st 12 noon—6 p.m.**
Banquet Saturday 7 p.m.	**$35**	**Saturday Nov. 22nd 9 a.m.—5 p.m.**

Special Convention room rates available at Doubletree Hotel, Santa Clara.
Telephone direct to Doubletree reservations by calling 800 528-0444 or 408 986-0700.
Telex reservation number is 668-309DTI. *Request special Forth Interest Group rates.*

___ Yes! I will attend the Forth Convention.
Number of pre-registered admissions _____ X $15 $ ____________
Number of banquet tickets _____ X $35 ____________
___ Yes! I want to join FIG and receive Forth Dimensions ($30 US, $43 foreign) ____________

TOTAL CHECK TO FIG $ ____________

Name ____
Address ____
Company ____
City ____________________ State ________ Zip ____________
Phone (_______) ____________________

Return to **Forth Interest Group**, P. O. Box 8231, San Jose, CA 95155 • **408 277-0668**

When New Becomes Obsolete

Dear Sir:

Stop, stop, for pity's sake, STOP! In anguished supplication I beg of you, whoever you may be: Please, not an 87-Standard Forth! The way things are going, in a few years you will have to change the name *Forth Dimensions* to *"Forth Fragmentations!"*

I concede that Forth-83 is not perfect, nor was fig-FORTH or Forth-79. But the situation is getting ridiculous with so many "official" versions of Forth. Is this the planned model change every four years that once made us love GM? I submit that no one will benefit from any further changes in "standard" Forth except those who would sell new Forth systems to the few unsuspecting but trusting souls who believe they must replace an "obsolete" system.

fig-FORTH was never so bad that it had to be junked; it is my firm conviction that [the change to] Forth-79 was cosmetic, and Forth-83 even more so. Why make it so difficult for fig-FORTH users to talk to Forth-79 or Forth-83 users? I think a much better solution to the problem of deficiencies in fig-FORTH would have been to develop a standard set of extension words for fig-FORTH which would have been added to existing systems at the discretion and convenience of the programmer.

It is still not too late! If we must live with the present standard Forth-83, then so be it. But let us not compound the felony by foisting another "standard" on the already troubled Forth community.

Yes, I do have an operating Forth-83 system. It is the excellent offering by Wilson Federici for the 6809; he is giving it away! However, I am seriously considering going back to my 6809 fig-FORTH, which I have been working on to use bank switching something like the 8088. I realize I can't make fig-FORTH behave exactly like Forth-83 by just adding additional words, but I hope to get close. However, I do know this — I will never update to a new standard Forth!

Let me address another point as related to standards. I have seen several letters lately asking for changes in Forth that would accommodate the thirty-two-bit crowd with their larger addressing space. Don't change Forth! Instead, add some new words which will process the extra addressing. What is wrong with words like **4DUP**, etc.? Additions like this would still make it possible for people like me, who plan to keep their eight- and sixteen-bit machines (like my 6809), to continue talking to the rest of the Forth community.

Perhaps I am in the wrong, though. Perhaps Forth is primarily used by professional programmers and system designers who simply tolerate the hobbyist crowd as long as they keep quiet. Am I a voice in the wilderness, talking only to myself? Am I the only hobbyist who programs in Forth because it is his favorite language? Oh, I have written

SHAREWARE

Fifth 2.0

- ***Tight Binding of Source and Code***
- ***Dictionary Editor - Allows Modification of Dictionary (browse, modify, reorder, recompile, insert, delete, ...)***
- ***Automatic Recompilation of Modified Source***
- ***Tree Structured Scoping***
- ***Full Screen Text Editor, 80 columns by N lines***
- ***Source Level Tracing and Breakpoints***
- ***Large Memory Model***
- ***32-bit Arithmetic***
- ***8087 Floating Point Support***
- ***MS-DOS File Support***
- ***Graphics Support***
- ***On Line Help Facility***
- ***Turnkey Applications***
- ***Produces Native Code***
- ***Includes F83 Public Domain Assembler***

For IBM or TI PC's with 128K, DOS 2.0 or better.
Amiga and Macintosh versions, First Quarter 1987.

Disk:	*$ 5.00*
Disk and Manual:	*$30.00*
Disk, Manual and Newsletter:	*$40.00*

CLICK Software
P.O. Box 10162
College Station, TX 77840
(409)-696-5432 or (409)-693-8853

Fifth 2.0 is a Shareware product, available on bulletin boards everywhere.

Amiga is a registered trademark of Commodore Corp.
MS-DOS is a registered trademark of Microsoft Corp.
TI is a registered trademark of Texas Instruments Corp.
Macintosh is a registered trademark of McIntosh Laboratory, Inc.
IBM is a registered trademark of International Business Machines Corp.

commercial programs for engineering and control, but in other languages (BASIC and Fortran) because the customer was afraid of a language "nobody ever heard of." Maybe a few more hobbyist users of Forth would actually make it easier to sell Forth programs.

Am I talking to the wrong group? Possibly FIG, too, is not interested in the hobbyist. Am I wasting my money belonging to an organization which appears to treat me with disdain? The questions are not strictly rhetorical; I hope someone will answer me. My concern about proliferating standards is related to program exchange among hobbyists. Already there is a problem in using much of the public-domain software because it is so often tied into 8088 machine language; do people think that the world begins and ends with IBM? More "standards" will only make matters worse.

I would like to close with one more question, "Who is kidding who?"

Sincerely,

R.D. (Doug) Lurie
Leominster, Massachusetts

Terminal TI

Dear Editor,

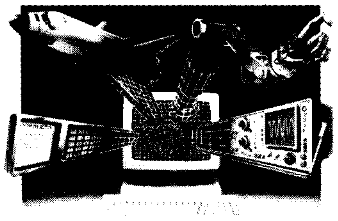
As a relatively recent convert to Forth (little more than a year of Forth), I was delighted to find that your publication is actively searching for articles on TI-Forth for the 99/4A. I write a column in our user-group newsletter (*MSP99*) on Forth.

With reference to "Simple Data Transfer Protocol" (*Forth Dimensions* VI/2), I submit the enclosed two pages. The first, second and last screens are my adaptation of the protocol to the 99/4A. The terminal emulator will work in half or full duplex. The routines are set up to use RS-232 port one at 300 baud. Data for alterations is given in the comments. Speeds greater than 1200 bps may require some routines to be in **CODE**. The only option required is the "-CRU" option, loaded in the first screen (that's what the "CLOAD" loads).

Cordially,
Glenn Davis
St. Paul, Minnesota

```
( serial communication constants    TI 99/4A  26SEP85  GED
  based on words from Forth Dimensions VI/2                    )
BASE->R DECIMAL 88 R->BASE CLOAD  STCR
BASE->R HEX
  04D0         CONSTANT  RATE      ( 04D0=300 bps; 01A1=1200 bps  )
   083         CONSTANT  PROTOCOL  ( 8 bits, no parity, 1 stop    )
 01300 2 /     CONSTANT  CARD      ( 1300=ports 1&2 1500=ports 3&4)
CARD    07 +  CONSTANT  LED       ( LED on RS232 card            )
CARD   020 +  CONSTANT  PORT   DECIMAL  ( 020=odd port 040=even )
PORT   13  +  CONSTANT  LDIR      ( LoaD Interval Register       )
PORT   16  +  CONSTANT  RTSON     ( Request To Send ON           )
PORT   18  +  CONSTANT  RIENB     ( Receive Interrupt ENaBle     )
PORT   21  +  CONSTANT  RBRL      ( Receive Buffer Reg Loaded    )
PORT   22  +  CONSTANT  XBRE      ( transmit Buffer Reg Empty    )
PORT   31  +  CONSTANT  RESET     ( RESET TMS9902 ACC chip       )
R->BASE -->



( serial communication primitives   TI 99/4A  13JUN85  GED
  based on words from Forth Dimensions VI/2                    )
BASE->R HEX
: XINIT   RESET SBO              ( reset serial port           )
    PROTOCOL 08 PORT LDCR        ( load rate, parity, stop bits )
    LDIR SBZ                     ( inhibit access to interval reg )
    RATE 0C PORT  LDCR ;         ( load rcv, xmt rate registers )
: ?XOUT    ( --- f )   XBRE TB ;
: ?XIN     ( --- f )   RBRL TB ;
: XOUT    ( char --- )
    LED SBO   RTSON SBO   BEGIN ?XOUT UNTIL
    08 PORT LDCR   RTSON SBZ   LED SBZ ;
: XIN     ( --- char )
  LED SBO  BEGIN ?XIN UNTIL  08 PORT STCR  RIENB SBZ  LED SBZ ;

R->BASE -->



( communication protocol primitives TI 99/4A  13JUN85  GED
  based on words from Forth Dimensions VI/2               )
BASE->R HEX

: TXCLR   0 XOUT ;
: ENQ
    BEGIN  5  ( enq ) XOUT  XIN  6 ( ack ) =  UNTIL ;
: ACK
    BEGIN  XIN  5 ( enq ) =   UNTIL
    6  ( ack )  XOUT ;
: 2DUP
    OVER OVER ;
: 2DROP
    DROP DROP ;

R->BASE -->



( transfer protocols primitives     TI 99/4A  13JUN85  GED
  based on words from Forth Dimensions VI/2 ) BASE->R DECIMAL
: SEND-BLOCK ( block-addr --- )
    DUP  B/BUF +  SWAP
    BEGIN  DUP C@  XOUT 1+  2DUP =
    UNTIL  2DROP ;
: TAKE-BLOCK ( block-addr --- )
    DUP  B/BUF +  SWAP
    BEGIN  XIN OVER C!  1+  2DUP =
    UNTIL  2DROP UPDATE ;
: SYNC  TXCLR
    BEGIN  5 XOUT
           ?XIN  IF  XIN 5 =  ELSE  0  THEN
    UNTIL  6 XOUT
    BEGIN  XIN 6 =  UNTIL ;
R->BASE -->
```

polyFORTH GETS YOUR PROGRAM FROM CONCEPT TO REALITY 4 TO 10 TIMES FASTER



THE ONLY INTEGRATED SOFTWARE DEVELOPMENT PACKAGE DESIGNED FOR REAL-TIME APPLICATIONS

If you're a real-time software developer, polyFORTH can be your best ally in getting your program up and running on time. In fact, on the average, you will develop a program 4 to 10 times faster than with traditional programming languages.

polyFORTH shortens development time by making the best use of your time. There are no long waits while you load editors, compilers, assemblers, and other tools, no long waits while they run—because everything you need is in a single, easy-to-use, 100% resident system. Using polyFORTH, you take a raw idea to fast, compiled code in seconds—and then test it interactively.

polyFORTH has everything you need to develop real-time applications: fast multi-tasking, multi-user OS; FORTH compiler, interpreters, and assemblers; editor and utilities; and over 400 primitives and debugging aids. With its unique modular structure, polyFORTH even helps you test and debug custom hardware interactively, and it is available for most 8, 16, and 32-bit computers.

FORTH, Inc. also provides its customers with such professional support services as custom application programming, polyFORTH programming courses, and the FORTH, Inc. "Hotline."

For more information and a free brochure, contact FORTH, Inc. today. FORTH, Inc., 111 N. Sepulveda Blvd., Manhattan Beach, CA 90266. Phone **(213) 372-8493.**

FORTH, Inc.

```
( transfer protocol user lexicons   TI 99/4A  13JUN85  GED
  based on words from Forth Dimensions VI/2 ) BASE->R DECIMAL
: XMT      ( first-scr# last-scr# --- )
    SYNC  1+ SWAP
    DO   ENQ  2 XOUT  I BLOCK SEND-BLOCK
    LOOP ENQ 4 ( eot ) XOUT ;
: RCV      ( first-scr# --- )
    SYNC
    BEGIN DUP BLOCK  ACK  XIN 2 ( stx ) OVER =
          IF   DROP TAKE-BLOCK  0
          ELSE 4 ( eot ) =
               IF   DROP FLUSH
               THEN 1
          THEN SWAP 1+ SWAP
    UNTIL DROP ;
R->BASE -->

( TI 99/4A terminal emulator                        26SEP85 GED   )
BASE->R HEX
0 VARIABLE ECHO
: TERMINAL XINIT
     BEGIN
       ?XIN  IF  XIN EMIT  ENDIF
       ?KEY -DUP IF
                    DUP 2 =  IF  QUIT   ENDIF
                    0837C C@ 020  AND
                      IF
                         ECHO @ IF  DUP EMIT  ENDIF
                         XOUT  ELSE  DROP
                      ENDIF
                  ENDIF
     AGAIN ;
R->BASE
```

Multiple LEAVEs by Relay

Dear Editor:

John Hayes' "Another Forth-83 LEAVE" (VII/1) stimulated me to try to find an even simpler way to handle multiple Forth-83 **LEAVE**s. I decided that a straightforward approach involved having each **LEAVE** simply branch to the next **LEAVE**, with the last one removing the index values from the return stack and branching to the word following **LOOP**.

This is accomplished by using a flag to show if there is a **LEAVE** in the loop; if so, **LOOP** sets up a forward branch. Also, if there is already a **LEAVE** present, an added **LEAVE** sets up a forward branch to the *new* **LEAVE**, not to the next word. Thus, when **LOOP** is reached, there cannot be more than one **LEAVE** whose branch jump needs resolution. In the code, **(DO)** and **(LOOP)** are the standard constructions, as in McCabe's *Forth Fundamentals*.

Sincerely,

Chester H. Page
Silver Spring, Maryland

```
: (LEAVE) R> R> DROP DROP BRANCH ;
: >RESOLVE ( addr---) HERE OVER - SWAP ! ;
: <RESOLVE ( addr---) HERE - , ;

0 VARIABLE LEAVE.FLAG

: DO 0 LEAVE.FLAG ! COMPILE (DO) HERE 3 ; IMMEDIATE
: LEAVE LEAVE.FLAG @ IF >RESOLVE THEN 1 LEAVE.FLAG !
   COMPILE (LEAVE) HERE 0 , ; IMMEDIATE
: LOOP 3 ?PAIRS COMPILE (LOOP) LEAVE.FLAG @
   IF >RESOLVE THEN <RESOLVE ; IMMEDIATE
```

Hackles and Hopes

Anyone who has read announcements of this year's FORML conference undoubtedly recalls the theme, "Extending Forth Towards the 87-Standard." Already this has raised both hackles and hopes among those who care about standardization work.

Some companies with huge Forth-79 programs chose not to switch to Forth-83 because of the in-house costs of upgrading functional systems. Some vendors felt it more important to provide a stable system to their existing clientele than to incorporate the changes required by Forth-83. And many assert that redefining words in the nucleus is unforgivable and potentially lethal to Forth's commercial viability.

How we view the Forth standard depends on who we are. System vendors have a commercial interest in product stability that hobbyists don't share. Professional users welcome functional improvements if they are of clear benefit, but resist energy-consuming changes unrelated to getting a good job done on time. Some language hackers embrace any opportunity to save a clock cycle and will modify their personal systems regularly without qualm.

The interests of its diverse users brought them to Forth in the first place, and meeting their needs perpetuates the language's use. Those needs are as real and specific as the measurable performance of a primitive word and must, as a matter of course, be part of the overall approach to the standards effort. It is evident that the Forth community has many interests, of which the most fundamental is success of the language itself. A standard arises from that common interest and so does resistance to changes. The intensity and volume of debate over Forth-83 obscured that common interest, creating an us/them, win/lose atmosphere. I hope anyone participating in future debates will have studied *Getting to Yes* by Roger Fisher and William Ury of the Harvard Negotiation Project.

This morning I spoke with Guy Kelly, chairman of the Forth Standards Team (FST), about the growing discussion over another Forth standard. He told me the FST has no plans for a Forth-87 and has not called any meetings on the topic. During two meetings leading to the Forth-83 Standard, the team had to deal with about five hundred proposals; at that time, members of the team decided there would be no future FST events without advance technical meetings at the regional level. Proposals would then be published and distributed to subscribers. Guy said that, as of his last poll, FST referees had not indicated any desire to begin this process. It is easy to see that, even if a new standard is proposed, 1987 is an unlikely target date for its completion.

Forth Dimensions maintains a neutral ground for discussion and accepts articles and letters to the editor related to this subject. We hope representatives of the standards team will write about its current plans and intentions, as well as the focus and likely restrictions of future FST meetings. Users and vendors are equally welcome to submit their viewpoints and suggestions. Our pages are limited, so items related to general topics (e.g., extension word sets, upward compatibility), overall direction and philosophy will carry the greatest weight; most experimental, technical proposals would be better placed at FORML. *Forth Dimensions* will attempt to take no sides, warning only that we are looking for more light than heat. Coherent discussion is welcome from all "sides" in the debate.

The proper place to send formal proposals is the Forth Standards Team (P.O. Box 4545, Mountain View, California 94040). A proposal (even one suggesting only that the Forth kernel remain unchanged) must be submitted in the manner established in the published Forth-83 Standard document, and which we will reproduce in these pages if space permits. Copies of the standard are available from several vendors and the Forth Interest Group.

As always, we will continue publishing material that should be instructive and useful to our readers right now. Enjoy the issue!

—Marlin Ouverson
Editor

Forth to the Future!

Forthmacs

for the Atari ST

32-bit Forth-83

- REAL stream files - no need for blocks
- Multi-window EMACS editor
- Interfaces to all ST graphics and system routines
- Multitasking
- Many programming tools
- Structured Decompiler
- Assembler/Disassembler
- Debugger
- Command editing, history, and completion
- Complete manual
- On-line documentation
- Newsletter

Simply the Best!

- Forthmacs + manual: $50
- With source: add $50

Bradley Forthware
P.O Box 4444
Mountain View, CA 94040

Forth Systems With a Segmented Memory Model

Richard Wilton
Marina del Rey, California

Most Forth interpreters confine themselves to a single logical address space in memory. Executable code, linked lists of addresses, and structures of initialized and uninitialized data are all stored within the same address space. Accessing any of these program elements within a Forth program is simple because any executable code or data item can be reached with a unique sixteen-bit address.

The lack of a structured "memory map" in Forth systems has not proven to be a major deficit in small computers with sixty-four Kbytes or less of main memory, or in computers without sophisticated operating systems. However, in complex microcomputer operating system environments, the use of a segmented memory model within a Forth interpreter offers clear advantages over older approaches.

Advantages of Memory Segmentation

The Intel 8086. Initially, the prevalence of computers based upon the Intel 8086 family of microprocessors provided the impetus for creating a Forth interpreter with a segmented memory model. The architecture of these processors is such that memory is most conveniently addressed in sixty-four Kbyte partitions or segments. Since the 8086 and related processors have four segment addressing registers, a software memory model which "maps" into these registers makes very effective use of memory.

For example, a memory model which uses four non-overlapping segments can conveniently utilize 256 Kbytes of RAM in an 8086-based machine, even though the software handles only sixteen-bit addresses for the most part. A Forth interpreter can thus make effective use of up to 256 Kbytes of RAM even with a sixteen-bit address interpreter.

Headerless and ROMable Code. There are other strong reasons for segmentation of memory in a Forth system apart from such hardware-related considerations. For example, applications developers who have no need for dictionary headers can easily excise them from a finished application if the headers are maintained in a separate memory partition. Also, the separation of executable code from other data greatly simplifies the implementation of a Forth interpreter which can be executed from read-only memory (ROM).

Links to High-Level Languages. Still another important reason for using a Forth interpreter with a segmented memory model is that most high-level language compilers generate executable modules with similar memory segmentation. It is easier to link to such "external" modules from a Forth system with a segmented memory structure.

The Basic Segmented Memory Model

The implementation of a Forth system with segmented memory is straightforward. The basic memory map consists of three discrete partitions or segments:

Code segment — contains all executable code

Data segment — contains bodies of colon definitions as well as initialized and uninitialized program data

Headers segment — contains names and addresses of all definitions

The parameter and return stacks may be placed in a fourth discrete segment. However, it is simpler in many respects to maintain the stacks in one of the other segments. Furthermore, certain operating systems constrain the stack to either the code or data segments.

Theoretically, there is no restriction on the amount of segmentation that can be done in creating the memory map. For instance, disk buffers, constants, or a "user" area for multitasking might each be mapped into separate segments of their own. However, the increased overhead involved in keeping track of many separate segments can degrade system performance as well as increase the complexity of the system.

The separation of executable code from data is particularly useful in environments in which memory protection schemes are used. For example, in its "protected" mode the Intel 80286 supports the creation of execute-only code segments and non-executable data segments; attempts to execute data or to modify existing code generate hardware exceptions which can trap to software error-handling routines. This feature might be much appreciated by Forth programmers whose programs would otherwise crash irretrievably due to such errors.

System Data Structures

The structures which contain the data upon which the interpretive Forth system operates reflect the segmentation of the system's memory map. The actual data structures used in a particular implementation necessarily depend upon the type of threaded code used.

Indirect-Threaded Code. For example, in an Indirect-Threaded Code (ITC) system, a colon definition is mapped into a list of sixteen-bit addresses in the data segment, as in Figure One.*

The code field contains the address of the inner interpreter's "nest" routine. Since this field points to executable code, the address refers to a location in the code segment. The body of the definition contains code field addresses (compilation addresses), all of which refer to locations in the data segment.

A **CODE** definition is mapped primarily into the code segment. The only part of a **CODE** definition to be found in the data segment is the address of the executable code, which is stored in the data segment at the definition's compilation address. (See Figure Two.)

Direct-Threaded Code. The structures are somewhat different in a Direct-Threaded Code (DTC) system. In a DTC implementation each colon definition references the inner interpreter's "nest" routine explicitly with a short fragment of executable code, as shown in Figure Three.

This executable fragment (a jump instruction) resides in the code segment. The "nest" routine uses the address stored in the code segment to find the body of the definition, which resides in the data segment. Similarly, a **CODE** definition resides entirely in the code segment.

Headers. The actual structure of dictionary headers, which are maintained in a segment of their own, greatly depends upon the structures used to represent Forth **VOCABULARY**s. However, the basic structure is simple, as in Figure Four.

The "compilation address" field deserves careful attention because the type of address it contains depends upon the implementation. In an ITC implementation, this address references a location in the data segment. In a DTC system, the address is located in the code segment. This difference is critical in understanding the function of a Forth system with a segmented memory map, although it is immaterial to a Forth applications programmer. (For example, the sequence ' <**name**> **EXECUTE** has the same effect in either implementation.)

Implications for System Implementation

Because code and data are separated, complex and unconventional colon definitions can be easily built. For example, the implementation of words containing **CREATE ... DOES>** or **;CODE** is very straightforward, especially in a DTC system where the size of the code fragment associated with a definition is not limited.

Since compilation addresses are maintained explicitly in dictionary headers, it is easy to implement a forward-referencing scheme. A forward reference is created with a "dummy" compilation address. Later, when the reference is resolved, the "dummy" address is replaced by a real one.

Both Unix and MS-DOS offer dynamic memory allocation facilities. Such dynamically-allocated memory is usually located in a partition outside of a requesting program's initial address space. A Forth which integrates a segmented memory model is easily adaptable to this scheme of memory allocation.

In Forth systems with partitioned memory, Forth memory operators such as @ and ! implicitly address the data segment. A problem arises when it is necessary to access other memory partitions explicitly. One simple approach is to utilize "long" memory operators which require both a "base address" (e.g., a segment address) and an "offset":

@L (base_addr offset -- value)

!L (value base_addr offset --)

CMOVEL (base_addr1 offset1 base_addr2 offset2 #bytes --)

This approach works well in practice. In any case, such "long" operations comprise only a small percentage of most applications.

The unique advantages of a Forth interpreter with a segmented memory model are not restricted to microprocessors which directly support memory segmentation. The segmented memory scheme works very well on the Motorola 68000, for example. Of course, for older eight-bit processors with only 64 Kbytes of address space, this sort of memory partitioning involves more bookkeeping overhead than it is worth. Nevertheless, a partitioned memory model in Z-80 or 6502 systems with bank-switched memory, such as the Apple IIe, may yet prove to be useful.

The performance of a Forth interpreter with a segmented memory map is not significantly affected by the separation of code, data and headers. Other factors, including the method of dictionary search, the nature of the threaded code implementation (e.g. ITC or DTC) and the efficiency of machine code primitives, influence performance to a much greater degree.

For example, execution speed benchmarks on an ITC implementation of an 8086-based Forth system (PC/Forth by Laboratory Microsystems) are virtually identical whether or not the memory map is segmented. Considering the improvement in efficiency and flexibility of memory utilization, a Forth system which incorporates a partitioned memory model is a better approach to Forth programming in demanding sixteen-bit systems with complex software environments.

***See figures on page 27.**

UBZ FORTH™

for the Amiga™

*FORTH-83 compatible
*32 bit stack
*Multi-tasking
*Separate headers
*Full screen editor
*Assembler
*Amiga DOS support
*Intuition support
*ROM kernel support
*Graphics and sound support
*Complete documentation
*Assembler source code included
*Monthly newsletter

$85

Shipping included
in continental U.S.
(Ga. residents add sales tax)

UBZ *Software*

(404)-948-4654

(call anytime)
or send check or money order to:

UBZ *Software*
395 St. Albans Court
Mableton, Ga. 30059

*Amiga is a trademark for Commodore Computer. UBZ FORTH is a trademark for UBZ Software.

An invitation to attend the eighth annual

FORML CONFERENCE

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

Following Thanksgiving
November 28 - 30, 1986

Asilomar Conference Center

Monterey Peninsula overlooking the Pacific Ocean
Pacific Grove, California

Theme: Extending Forth towards the 87-Standard

FORML isn't part of the Standards Team, but the conference is an opportunity to present your ideas for additions to the Forth standard. Papers are also welcome on other Forth topics. Meet other Forth professionals and learn about the state of the art in Forth applications, techniques, and directions.

To get your registration packet call the FIG Business Office (408) 277-0668 or write to: FORML Registration, Forth Interest Group, P. O. Box 8231, San Jose, CA 95155.

Registration:	$275	Double Room
	$325	Single Room (Limited availability)
	$150	Non-conference guest (Share a double room)

Registration includes room, meals, conference materials, and social events.

Space is limited, advance registration is required.

FORML

The Point Editor

J. Brooks Breeden
Columbus, Ohio

Hardin's Law: You can never do just one thing.

Often, I find books and back issues of *Forth Dimensions* contain tips and techniques that I didn't understand or need when I first read them but that now make perfect sense and are exactly what I'm looking for. Sometimes, however, what I need to help me solve a problem is hiding, disguised as a solution to a different kind of problem. For example:

Recently, while developing a computer-assisted instruction (CAI) drill-and-practice test on storm drainage, I wanted to draw a map of the United States, showing zones of approximately equal rainfall. Because I wanted just one small map in a corner of the display, I found an 8x10-inch map of the USA and traced it onto graph paper, redrawing the outline with straight line segments that gave a reasonable facsimile of the USA's shape. Next, by trial and error, I found a scale which related the USA's vertical dimension to the y-axis coordinate range I wanted on the display, and scaled the x,y coordinates based on 0,0 in the upper-left corner of Washington state. The USA outline required the first 128 x,y coordinate pairs shown in Figure One. The remainder of the points in Figure One are used to draw the lines of "approximately equal rainfall."

I use PC/Forth 3.1 from Laboratory Microsystems, Inc. and, normally, CAI graphics are simple to program using the supplied routines for **LINE**, **FLOOD**, **ARC**, etc. I was worried about running out of memory with this application, however, and I didn't want to compile a word consisting of *x y x y* **LINE** 128 times for the USA's outline! I had also measured the x and y coordinates using the same scale, and the display required an x-axis multiplier because the pixels on IBM displays aren't square. Having gotten this far, I finally stopped to think about the problem. How was I going to draw the USA?

Three methods came to mind:

1. Define a command to **DRAW** from the last point to the next point, letting the computer re-scale the x-axis and add an offset from an "x,y origin" on the display. I could use a brute-force approach with screens full of **X Y DRAW** instead of **X Y X Y LINE** (after all, it was just one map), and I could relocate the map on the display by changing the coordinates of the origin.
2. Create the **DRAW** command as above, manually re-scale the x-axis and , (comma) the coordinates into an array. A loop could read the array and draw the USA's outline.
3. Recall a previously stored image from disk with PC/Forth's **GET-IMAGE** and overlay it with the rest of the tutorial.

The first method would require several screens of **X Y DRAW** and matching a point would require looking through literally hundreds of numbers and **DRAW**s. The second method would take fewer screens, but editing numbers separated by only commas (no **DRAW**s) would be even harder. Using , (comma) to lay down all the points in the dictionary would use lots of memory, too. The third method wouldn't solve anything, since I had to draw the image, somehow, before I could store it!

BASIC draws points from arrays stored in disk files all the time! Then, I rememberd an exercise in *Starting Forth* (chapter ten, exercise five, page 288) to write words to accept pairs of numbers from the keyboard, write them to a block, then recall and format them on the display. Eureka! I could store coordinate pairs in a block and draw the USA's outline by using a loop to read coordinates from the block.

Hardin's Law Strikes

Have you ever entered 256 consecutive numbers? Did you ever make a mistake? Don't answer! It's not enough to just write numbers to a block and read them back. You have to be able to change a number and you need to be able to add or delete one, or several (in the middle, yet). You see, you can never do just one thing! I needed a "point editor."

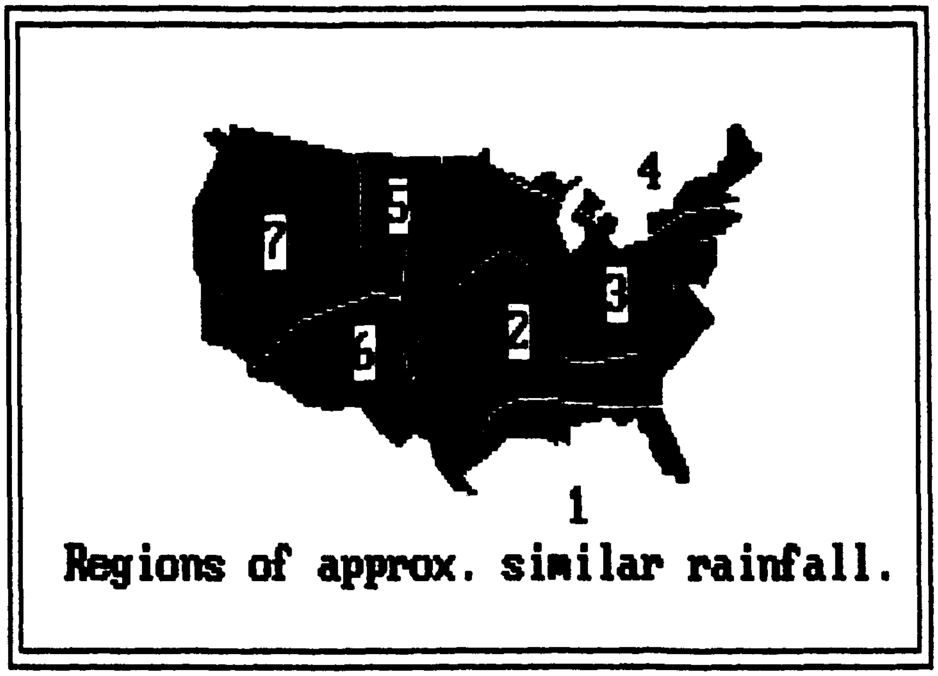


Figure Two

SOFTWARE for the HARDCORE

MasterFORTH

FORTH-83 STANDARD

- • 6809 Systems available for FLEX disk sustems $150
OS9/6809 . . . . $150
- • 680x0 Systems available for MACINTOSH $125
CP/M-68K $150
- • tFORTH/20 for 68020 Single Board Computer

Disk based development system under OS9/68K . . . $290

EpROM set for complete stand-alone SBC $390
- • Forth Model Library - List handler, spreadsheet, Automatic structure charts . . . each . $40
- Target compilers : 6809, 6801, 6303, 680x0, 8088, Z80, 6502

Talbot Microsystems
1927 Curtis Ave
Redondo Beach
CA 90278
(213) 376-9941

HARDWARE for the HARDCORE

68020 SBC, 5 1/4" floppy size board with 2MB RAM, 4 x 64K EpROM sockets, 4 RS232 ports, Centronics parallel port, timer, battery backed date/time, interface to 2 5 1/4" floppies and a SASI interface to 2 winchester disks $2750

68881 flt pt option $500

OS9 multitask&user OS. . $350

FAST! int. benchmarks speeds are 2 x a VAX780, 10 x an IBM PC

POINTED.SCR

POINTED ("Point Ed-itor") stores points in a block. Using a non-DOS, block-oriented Forth like polyFORTH, that's it; you have a block of points. If, however, your Forth uses screens in a DOS file, will the point data be a screen in your application's file, or a separate file on the disk? Because I wanted to be able to read the point data from a PC/Forth **TURNKEY** application (a DOS .COM file which wouldn't *have* any blocks), I needed to store the points in a separate, named, "point file" on the disk. The DOS interface was something I hadn't intended to deal with: how and when to open and close screen files within Forth, from a .COM file, FCBs, handles, etc. Hardin's Law.

To put off dealing with DOS for awhile, I chose to use **POINTED**'s screen ten as a data block for "editing." I entered all the required points and used LMI's **SCOPY** utility both to create a new file and to transfer **POINTED**'s screen ten to the new file's screen zero. (Screen zero can be read and written to, but cannot be loaded; a distinct advantage!)

Screens one through four contain the "command-driven" program. Defined words allow scaling the x-axis, entering, changing, deleting, listing and printing points stored in block ten. Screens five through seven make a simple menu-driven interface that a graduate student assistant not familiar with Forth can use to input points (Hardin's Law ... Murphy's Law, too). I like to think the source for these screens is self documenting.

Screen eight contains words for re-scaling between "raw" data and CGA/EGA scaled data. These words aren't normally loaded, as they can be dangerous, although I suppose the *most* dangerous word is an option on the menu. The "best" scale factor for CGA/EGA will vary with the vertical adjustment of your display. (Yours will probably be less.) Experiment, but backup block ten before you mess around — it can be frustrating to lose the original data and have to re-enter it.

A more sophisticated version would create a file and work directly with its screen zero without **POINTED**'s screen ten as the "middle man." Another improvement would automatically up-

```
Points list for screen 10
Pt.#    X    Y
#  0    4    1    #  1    6    2    #  2    6    0    #  3   14    2    #  4   24    4
#  5   38    5    #  6   44    5    #  7   46    3    #  8   46    3    #  9   46    6
# 10   52    8    # 11   50    9    # 12   52    8    # 13   52    9    # 14   52    9
# 15   54    7    # 16   56    7    # 17   56    8    # 18   58    9    # 19   60    7
# 20   60    8    # 21   60    9    # 22   58   10    # 23   56   14    # 24   58   19
# 25   58   19    # 26   60   17    # 27   60   15    # 28   58   14    # 29   60   11
# 30   60   11    # 31   60    9    # 32   62   10    # 33   64   12    # 34   62   13
# 35   62   14    # 36   64   13    # 37   66   14    # 38   64   17    # 39   66   18
# 40   66   18    # 41   70   16    # 42   70   14    # 43   70   13    # 44   74   12
# 45   74   11    # 46   76    8    # 47   80    7    # 48   80    6    # 49   82    5
# 50   82    1    # 51   82    1    # 52   84    0    # 53   84    1    # 54   86   -4
# 55   86    4    # 56   88    5    # 57   84    8    # 58   82   10    # 59   84   11
# 60   82   12    # 61   84   13    # 62   84   14    # 63   82   15    # 64   82   16
# 65   80   17    # 66   80   19    # 67   80   20    # 68   78   24    # 69   78   24
# 70   78   23    # 71   76   23    # 72   80   28    # 73   78   30    # 74   78   30
# 75   76   32    # 76   76   32    # 77   72   36    # 78   72   38    # 79   72   41
# 80   76   49    # 81   76   51    # 82   74   51    # 83   74   50    # 84   72   50
# 85   70   46    # 86   70   45    # 87   68   43    # 88   68   42    # 89   66   43
# 90   64   42    # 91   58   43    # 92   58   44    # 93   58   45    # 94   58   46
# 95   56   45    # 96   56   46    # 97   54   45    # 98   48   45    # 99   44   48
#100   42   50    #101   44   53    #102   40   52    #103   38   49    #104   36   45
#105   34   44    #106   32   46    #107   30   45    #108   28   43    #109   26   40
#110   24   40    #111   22   41    #112   18   40    #113   12   37    #114    8   36
#115    8   34    #116    6   32    #117    2   31    #118    2   27    #119    2   23
#120    0   21    #121    0   17    #122    0   15    #123    0   12    #124    2   10
#125    4    4    #126    2    2    #127    4    1    #128    4    1    #129 32898224
#130    6   38    #131   12   32    #132   16   28    #133   28   24    #134   24    2
#135   26    7    #136   26   13    #137   26   21    #138   28   24    #139   32   25
#140   40    3    #141   34   13    #142   32   25    #143   32   31    #144   34   37
#145   50    4    #146   48    7    #147   50    9    #148   60   13    #149   64   16
#150   70   17    #151   76   13    #152   86   11    #153   34   45    #154   34   37
#155   40   24    #156   44   20    #157   48   18    #158   52   21    #159   56   32
#160   58   34    #161   62   34    #162   68   33    #163   78   17    #164   86   15
#165   44   51    #166   42   48    #167   44   44    #168   46   41    #169   50   39
#170   54   39    #171   64   40    #172   74   40
```

Figure One Original points scaled from map.

date the number of points in the file, eliminating the hassle of manually keeping track of the number of points. Still, if you store sets of points to draw different things in the same block, such as the "lines of approximately equal rainfall," you will have to keep track of starting and ending point numbers anyway. There are *many* other improvements to **POINTED** that I leave to the reader to implement; but remember, I wasn't trying to write a universal point editor. I was trying to draw just one map!

Screen nine contains the drawing routines. This screen is not a part of POINTED.SCR, per se, but is included in the file for convenience. Normally, you copy it to your application and modify the file names, origin, etc., as required. These definitions should be self-explanatory with the exception of **V+** which adds vectors. In **DRAW-PTS**, **PT#** fetches a point's coordinate pair and the **ORIGIN**'s coordinate pair. **V+** adds the **ORIGIN**'s x to the point's x and the **ORIGIN**'s y to the point's y. By changing the origin, you can draw the same image at different places on the screen. This allows "fine tuning" the display. Note that the file names compiled by **DRAWER** and **POINTFILE** have extensions. If you **TURNKEY** the application, be sure the extension for the name in **DRAWER** is .COM.

Listing One

Screen # 0

```
( POINTED  Point Editor for drawings        jbb 10:46 04/28/86 )
( Last change:   Screen  002                jbb 15:31 04/29/86 )

              Copyright 1986, by J. Brooks Breeden
                 All commercial rights reserved.
You may freely use this program for personal non-commercial use.

The point editor creates files of binary points (coordinates)
 stored in Screen 10 of THIS file.  Up to 224 PAIRS of x y
 values may be stored.  Use SCOPY to create a new file and
 transfer data in Screen 10 to the new file's Screen 0.
 DRAWER.SCR shows how to call the points from another program.

This program is written in and for the Forth-83 implementation
        PC/FORTH 3.1 by Laboratory Microsystems, Inc.
```

Screen # 1

```
( Scaling? points                            jbb 10:49 04/28/86 )
10 CONSTANT POINTS           ( block 10 is used to store points)

\ step past possible date stamp on line 0, & offset into block
: POINTSBLOCK ( i - adr)   2* 2*  POINTS BLOCK + 64 + ;
: !POINTS ( n n i - )   POINTSBLOCK 2! UPDATE ;
VARIABLE ?SCALE             ( flag to scale x-axis: -1=yes,0=no )
: SCALEIT?  CLS  ." Scale x-axis?  1 = yes, 0 = no. "
   #IN  ?SCALE ! ;                     ( #IN gets integer input)
\ This is for EGA, substitute 12 5 */ for CGA display scaling.
: SCALING-X? ( x y - x' y)  ?SCALE @ IF SWAP
   13 10 */ SWAP THEN ;
-->
```

Screen # 2

```
( Enter points                              jbb 15:31 04/29/86 )
: ENTER-PT ( - x y)   CR ." Enter x <CR> y <CR>. " CR
   ." X = " #IN CR  ." Y = " #IN  CR ;
: WAIT  28 24 GOTOXY ." Press any key to continue."  KEY DROP ;
: ?+PTS ( - start count)  ." Starting at which point # ? "
   CR #IN  224 MIN  CR ." How many points? " #IN  224 MIN ;
: ?-PTS ( - start end)   ." Delete from point # ? " CR  #IN
   224 MIN  CR ." Delete thru point # ? " CR #IN  224 MIN ;
-->
```

Screen # 3

```
( Change & delete pts.                       jbb 16:41 04/27/86 )
: +PTS ( start count - )   SWAP POINTSBLOCK SWAP 2* 2* DUP >R
   OVER + R>  CMOVE> UPDATE   ;
: -PTS ( 1st last - )   1+ 224 OVER - 2* 2* >R
   POINTSBLOCK SWAP POINTSBLOCK R>  CMOVE  UPDATE  ;

: CHANGE-PT   CLS SCALEIT?  CR ." Enter the point #. " CR #IN
   ENTER-PT  SCALING-X?  ROT !POINTS UPDATE SAVE-BUFFERS ;
: ADDPTS ( start count - )  CLS  SCALEIT?  CR  OVER + SWAP
   DO ." Pt. # " I .  ENTER-PT  SCALING-X?  I !POINTS  LOOP ;
: INSERTPTS   CLS ?+PTS  2DUP  +PTS ADDPTS  SAVE-BUFFERS ;
: DELETEPTS   CLS ?-PTS  -PTS SAVE-BUFFERS ;
-->
```

Screen # 4

```
( Show & print pts                          jbb 10:46 04/28/86 )
: .PTS ( start count - )  ." Points list for screen 10" CR
   ." Pt.#   X   Y "   OVER + SWAP
   DO  I POINTSBLOCK 2@  SWAP I 5 MOD 0=
     IF  CR  ELSE  3 SPACES  THEN
  ." #" I 3 U.R  SPACE  4 U.R 4 U.R  LOOP  CR ;

: SHOWPTS    CLS ?+PTS  CLS .PTS WAIT ;
: PRINTPTS   CLS ?+PTS PRINTER .PTS CONSOLE ;

: WIPE-BLOCK-10 ( - )   POINTS BLOCK 1024 BLANK ;
( DANGER!!!   If you do this you wipe it ALL away!!!)
-->
```

Screen # 5

```
( Choices                                    jbb 10:47 04/28/86 )
: CHOICES   BLUE BG CLS WHITE FG 35 7 GOTOXY ." MAIN MENU"
   GRAY FG
   32 11 GOTOXY  ." (S)how points"
   32 12 GOTOXY  ." (I)nsert points"
   32 13 GOTOXY  ." (C)hange a point"
   32 14 GOTOXY  ." (D)elete points"
   32 15 GOTOXY  ." (P)rint points"
   32 16 GOTOXY  ." (W)ipe block 10!!! ( dangerous!!!)"
   32 18 GOTOXY  ." Esc leaves the program"  WHITE FG
   26 20 GOTOXY  ." Press the first letter of your choice."
   GRAY FG ;
-->
```

DRAWER.SCR

The second listing, DRAWER.SCR is a demo to show that **DRAWER** does indeed draw points entered with **POINTED** and that the result does resemble the USA. Screens one through eight build a dummy of the application that started the whole thing. An EGA screen dump of the display is shown in Figure Two. It's ironic that the source to dummy the application is almost longer than everything else. I could not avoid the DOS interface forever. Screen five shows how I sequenced the opening and closing of files so the program doesn't get lost in DOS and crash, leaving files lying around open.

The application uses EGA graphics and floating-point math extensions. The demo dummy lacks about thirty screens of irrelevant, generic CAI question drivers, answer checkers and random problem generators. Everything *looks* exactly like the real McCoy, except it doesn't work. You do *not* need floating-point math routines to load and run the demo, but you *do* need an EGA card and display to run the demo as it is written. (Attempting to run the demo from PC/Forth without loading **EGAGRAPH** first will result in a truly spectacular hard crash!) To run with a CGA display, eliminate color references, re-scale the points and adjust all y-axis references in DRAWER.SCR accordingly.

I hope some of the "other things" I had to do in trying to draw just one map may be of help to others trying to do just one thing, sometime. If anyone is interested in the inner workings of the real application, please contact me. I'd be happy to discuss the specifics.

```
Screen # 6
( Message                                jbb 10:47 04/28/86 )
: MESSAGE   CLS  12 10 GOTOXY
  ." Note: Points will be stored in Screen 10 of this file."
  29 14 GOTOXY ." Be certain Screen 10 is wiped or that"
  29 15 GOTOXY ." it is the point-set you wish to edit."
  29 16 GOTOXY ." Use SCOPY to make a new file and to copy "
  29 17 GOTOXY ." Screen 10 to Screen 0 of the new file." ;
-->
```

```
Screen # 7
( Main program                           jbb 10:47 04/28/86 )
: MENU   CHOICES  KEY
   DUP       27 = IF DROP EXIT        ELSE
   DUP ASCII S = IF DROP SHOWPTS     ELSE
   DUP ASCII I = IF DROP INSERTPTS  ELSE
   DUP ASCII C = IF DROP CHANGE-PT  ELSE
   DUP ASCII D = IF DROP DELETEPTS  ELSE
   DUP ASCII P = IF DROP PRINTPTS   ELSE
   DUP ASCII W = IF DROP WIPE-BLOCK-10  ELSE
   DROP 500 50 BEEP  THEN
   THEN THEN THEN THEN THEN THEN RECURSE ;

: POINTED   MESSAGE WAIT  MENU ;
```

```
Screen # 8
( Optional dangerous words               jbb 15:24 04/29/86 )
\ There is obviously rounding error in these words.
: UNSCALE-CGA ( - ) 173 0    ( convert CGA scaling to raw data)
   DO  I POINTSBLOCK 2@  SWAP 5 12 */ SWAP I  !POINTS
   LOOP ;
: UNSCALE-EGA ( - ) 173 0    ( convert EGA scaling to raw data)
   DO  I POINTSBLOCK 2@  SWAP 10 13  */ SWAP I  !POINTS
   LOOP ;
: SCALE-CGA ( - ) 173 0       ( convert raw data to cga scaling)
   DO  I POINTSBLOCK 2@  SWAP 12 5 */ SWAP I  !POINTS
   LOOP ;
: SCALE-EGA ( - ) 173 0       ( convert raw data to ega scaling)
   DO  I POINTSBLOCK 2@  7 4 */ SWAP 7 4 */
     13 10 */ SWAP I  !POINTS
   LOOP ;
```

```
Screen # 9
( Drawer routines                        jbb 15:30 04/29/86 )
\ Copy this block into your application and modify as needed.
2VARIABLE CP             ( holds coordinates of graphic cursor)
: AT ( x y - )   CP 2! ;           ( set graphic cursor "AT" CP)
: DRAW ( x y - )   2DUP CP 2@ LINE AT ; ( draw line, update CP)
: V+ ( x y x1 y1 - x2 y2)   >R ROT + SWAP R> + ; ( add vectors)
\ Be sure to change extension of DRAWER to .COM if you TURNKEY
: DRAWER    " DRAWER.SCR " ;        ( the file-name to return to)
: POINTFILE " USAEGA.SCR " ;   ( the file-name holding points )
0 CONSTANT POINTS                          ( screen 0 in pointfile )
380 66 2CONSTANT ORIGIN       ( upper left corner xy of drawing)
\ Note: the POINTSBLOCK defined in Screen 1 is not the same.
: POINTSBLOCK ( n -)   POINTS BLOCK + 64 + ;     ( step +1 line)
: PT# ( n - x y )   2* 2* POINTSBLOCK 2@ ORIGIN V+ ;
: DRAW-PTS ( hi lo - )   DO  I PT# DRAW  LOOP ;
```

Continued on page 23.

FORTH INTEREST GROUP MAIL ORDER FORM

P.O. Box 8231   San Jose, CA 95155   (408) 277-0668

MEMBERSHIP

IN THE FORTH INTEREST GROUP

108 - MEMBERSHIP in the FORTH INTEREST GROUP & Volume 8 of FORTH DIMENSIONS. No sales tax, handling fee or discount on membership. See the back page of this order form.

The Forth Interest Group is a worldwide non-profit member-supported organization with over 4,000 members and 90 chapters. FIG membership includes a subscription to the bi-monthly publication, FORTH Dimensions. FIG also offers its members publication discounts, group health and life insurance, an on-line data base, a large selection of Forth literature, and many other services. Cost is $30.00 per year for USA, Canada & Mexico; all other countries may select surface ($37.00) or air ($43.00) delivery.

The annual membership dues are based on the membership year, which runs from May 1 to April 30.

When you join, you will receive issues that have already been circulated for the current volume of Forth Dimensions and subsequent issues will be mailed to you as they are published.

You will also receive a membership card and number which entitles you to a 10% discount on publications from FIG. Your member number will be required to receive the discount, so keep it handy.

HOW TO USE THIS FORM

1. Each item you wish to order lists three different Price categories:

   Column 1 - USA, Canada, Mexico
   Column 2 - Foreign Surface Mail
   Column 3 - Foreign Air Mail

2. Select the item and note your price in the space provided.

3. After completing your selections enter your order on the fourth page of this form.

4. Detach the form and return it with your payment to the **Forth Interest Group**.

FORTH DIMENSIONS BACK VOLUMES

The six issues of the volume year (May — April)

101 - Volume 1 FORTH Dimensions (1979/80)$15/16/18 ______
102 - Volume 2 FORTH Dimensions (1980/81)$15/16/18 ______
103 - Volume 3 FORTH Dimensions (1981/82)$15/16/18 ______
104 - Volume 4 FORTH Dimensions (1982/83)$15/16/18 ______
105 - Volume 6 FORTH Dimensions (1983/84)$15/16/18 ______
106 - Volume 5 FORTH Dimensions (1984/85)$15/16/18 ______
107 - Volume 7 FORTH Dimensions (1985/86)$20/21/24 ______

FORML CONFERENCE PROCEEDINGS

FORML PROCEEDINGS — FORML (the Forth Modification Laboratory) is an informal forum for sharing and discussing new or unproven proposals intended to benefit Forth. Proceedings are a compilation of papers and abstracts presented at the annual conference. FORML is part of the Forth Interest Group.

310 - FORML PROCEEDINGS 1980 . . . . $30/33/40 ________
Technical papers on the Forth language and extensions.

311 - FORML PROCEEDINGS 1981 . . . . $45/48/55 ________
Nucleus layer, interactive layer, extensible layer, metacompilation, system development, file systems, other languages, other operating systems, applications and abstracts without papers.

312 - FORML PROCEEDINGS 1982 . . . . $30/33/40 ________
Forth machine topics, implementation topics, vectored execution, system development, file systems and languages, applications.

313 - FORML PROCEEDINGS 1983 . . . . $30/33/40 ________
Forth in hardware, Forth implementations, future strategy, programming techniques, arithmetic & floating point, file systems, coding conventions, functional programming applications.

314 - FORML PROCEEDINGS 1984 . . . . $30/33/40 ________
Expert systems in Forth, using Forth, philosophy, implementing Forth systems, new directions for Forth, interfacing Forth to operating systems, Forth systems techniques, adding local variables to Forth.

315 - FORML PROCEEDINGS 1985 . . . . $35/38/45 ________
Also includes papers from the 1985 euroFORML Conference. Applications: expert systems, data collection, networks. Languages: LISP, LOGO, Prolog, BNF. Style: coding conventions, phrasing. Software Tools: decompilers, structure charts. Forth internals: Forth computers, floating point, interrupts, mulitasking, error handling.

BOOKS ABOUT FORTH

200 - ALL ABOUT FORTH .... $25/26/35 ________
Glen B. Haydon
An annotated glossary for MVP Forth; a 79-Standard Forth.

216 - DESIGNING & PROGRAMMING
N PERSONAL EXPERT SYSTEMS .. $19/20/29 ________
E Carl Townsend & Dennis Feucht
W Introductory explanation of AI-Expert System Concepts. Create your own expert system in Forth. Written in 83-Standard.

217 - F83 SOURCE .... $25/26/35 ________
N Henry Laxen & Michael Perry
E A complete listing of F83 including source and shadow
W screens. Includes introduction on getting started.

218 - FOOTSTEPS IN AN EMPTY VALLEY
N (NC4000 Single Chip Forth Engine) $25/26/35 ________
E Dr. C. H. Ting
W A thorough examination and explanation of the NC4000 Forth chip including the complete source to cmForth from Charles Moore.

219 - FORTH: A TEXT AND REFERENCE $25/26/35 ________
N Mahlon G. Kelly & Nicholas Spies
E A text book approach to Forth with comprehensive referen-
W ces to MMS Forth and the 79 and 83 Forth Standards.

220 - FORTH ENCYCLOPEDIA .... $25/26/35 ________
Mitch Derick & Linda Baker
A detailed look at each fig-Forth instruction.

225 - FORTH FUNDAMENTALS, V.1 ... $16/17/20 ________
Kevin McCabe
A textbook approach to 79-Standard Forth

230 - FORTH FUNDAMENTALS, V.2 ... $13/14/18 ________
Kevin McCabe
A glossary.

232 - FORTH NOTEBOOK .... $25/26/35 ________
Dr. C. H. Ting
Good examples and applications. Great learning aid. PolyFORTH is the dialect used. Some conversion advice is included. Code is well documented.

233 - FORTH TOOLS .... $22/23/32 ________
Gary Feierbach & Paul Thomas
The standard tools required to create and debug Forth-based applications.

235 - INSIDE F-83 .... $25/26/35 ________
Dr. C. H. Ting
Invaluable for those using F-83.

237 - LEARNING FORTH .... $17/18/27 ________
Margaret A. Armstrong
Interactive text, introduction to the basic concepts of Forth. Includes section on how to teach children Forth.

240 - MASTERING FORTH .... $18/19/22 ________
Anita Anderson & Martin Tracy
A step-by-step tutorial including each of the commands of the Forth-83 International Standard; with utilities, extensions and numerous examples.

245 - STARTING FORTH (soft cover) ... $22/23/32 ________
Leo Brodie
A lively and highly readable intruduction with exercises.

246 - STARTING FORTH (hard cover) .. $24/25/29 ________
Leo Brodie

255 - THINKING FORTH (soft cover) ... $16/17/20 ________
Leo Brodie
The sequel to "Starting Forth". An intermediate text on style and form.

265 - THREADED INTERPRETIVE LANGUAGES ... $25/26/35
R. G. Loelinger ________
Step-by-step development of a non-standard Z-80 Forth.

270 - UNDERSTANDING FORTH .... $3.50/5/6 ________
Joseph Reymann
A brief introduction to Forth and overview of its structure.

ROCHESTER PROCEEDINGS

The Institute for Applied Forth Research, Inc. is a non-profit organization which supports and promotes the application of Forth. It sponsors the annual Rochester Forth Conference.

321 - ROCHESTER 1981
(Standards Conference) .... $25/28/35 ________
79-Standard, implementing Forth, data structures, vocabularies, applications and working group reports.

322 - ROCHESTER 1982
(Data bases & Process Control) ... $25/28/35 ________
Machine independence, project management, data structures, mathematics and working group reports.

323 - ROCHESTER 1983
(Forth Applications) .... $25/28/35 ________
Forth in robotics, graphics, high-speed data acquisition, real-time problems, file management, Forth-like languages, new techniques for implementing Forth and working group reports.

324 - ROCHESTER 1984
(Forth Applications) .... $25/28/35 ________
Forth in image analysis, operating systems, Forth chips, functional programming, real-time applications, cross-compilation, multi-tasking, new techniques and working group reports.

325 - ROCHESTER 1985
(Software Management & Engineering) $20/21/24 ______
Improving software productivity, using Forth in a space shuttle experiment, automation of an airport, development of MAGIC/L, and a Forth-based business applications language; includes working group reports.

THE JOURNAL OF FORTH APPLICATION & RESEARCH

A refereed technical journal published by the Institute for Applied Forth Research, Inc.

403 - JOURNAL OF FORTH RESEARCH V.2 #1
Forth Machines. .... $15/16/18 ________

404 - JOURNAL OF FORTH RESEARCH V.2 #2
Real-Time Systems. .... $15/16/18 ________

405 - JOURNAL OF FORTH RESEARCH V.2 #3
Enhancing Forth. .... $15/16/18 ________

406 - JOURNAL OF FORTH RESEARCH V.2 #4
Extended Addressing. .... $15/16/18 ________

407 - JOURNAL OF FORTH RESEARCH V.3 #1
Forth-based laboratory systems and data structures.
.... $15/16/18 ________

REPRINTS

420 - BYTE REPRINTS .... $5/6/7 ________
Eleven Forth articles and letters to the editor that have appeared in *Byte* Magazine.

DR. DOBB'S JOURNAL

This magazine produces an annual special Forth issue which includes source-code listing for various Forth applications.

422 - DR. DOBB'S 9/82	$5/6/7	________
423 - DR. DOBB'S 9/83	$5/6/7	________
424 - DR. DOBB'S 9/84	$5/6/7	________
425 - DR. DOBB'S 10/85	$5/6/7	________
426 - DR. DOBB'S 7/86	$5/6/7	________

HISTORICAL DOCUMENTS

501 - KITT PEAK PRIMER . . . . $25/27/35 ________
One of the first institutional books on Forth. Of historical interest.

502 - Fig-FORTH INSTALLATION MANUAL $15/16/18 ________
Glossary model editor — We recommend you purchase this manual when purchasing the source-code listing.

503 - USING FORTH . . . . $20/21/23 ________
FORTH, Inc.

REFERENCE

305 - FORTH 83-STANDARD . . . . $15/16/18 ________
The autoritative description of 83-Standard Forth. For reference, not instruction.

300 - FORTH 79-STANDARD . . . . $15/16/18 ________
The authoritative description of 79-Standard Forth. Of historical interest.

ASSEMBLY LANGUAGE SOURCE CODE LISTINGS

Assembly Language Source Listings of fig-Forth for Specific CPUs and machines with compiler security and variable length names.

514 - 6502/SEPT 80	$15/16/18	________
515 - 6800/MAY 79	$15/16/18	________
516 - 6809/JUNE 80	$15/16/18	________
517 - 8080/SEPT 79	$15/16/18	________
518 - 8086/88/MARCH 81	$15/16/18	________
519 - 9900/MARCH 81	$15/16/18	________
521 - APPLE II/AUG 81	$15/16/18	________
523 - IBM-PC/MARCH 84	$15/16/18	________
526 - PDP-11/JAN 80	$15/16/18	________
527 - VAX/OCT 82	$15/16/18	________
528 - Z80/SEPT 82	$15/16/18	________

MISCELLANEOUS

601 - T-SHIRT SIZE ________________
Small, Medium, Large and Extra-Large.
White design on a dark blue shirt. . $10/11/12 ________

602 - POSTER (BYTE Cover) . . . . $5/6/7 ________

616 - HANDY REFERENCE CARD . . . . FREE ________

683 - FORTH-83 HANDY REFERENCE CARD . . FREE ______

FORTH MODEL LIBRARY

The model applications disks described below are new additions to the Forth Interest Group's library. These disks are the first releases of new professionally developed Forth applications disks. Prepared on 5 1/4" disks, they are IBM MSDOS 2.0 and up compatible. The disks are compatible with Forth-83 systems currently available from several Forth vendors. Macintosh 3 1/2" disks are available for MasterFORTH systems only.

Forth-83 Compatibility IBM MSDOS

Laxen/Perry F83	LMI PC/FORTH 3.0
MasterFORTH 1.0	TaskFORTH 1.0
PolyFORTH® II	

Forth-83 Compatibility Macintosh
MasterFORTH

ORDERING INFORMATION

701 - A FORTH LIST HANDLER V.1 . . . . $40/43/45 ________
by Martin J. Tracy
Forth is extended with list primitives to provide a flexible high-speed environment for artificial intelligence. ELISA and Winston & Horn's micro-LISP are included as examples. Documentation is included on the disk.

702 - A FORTH SPREADSHEET V.2 . . . . $40/43/45 ________
by Craig A. Lindley
This model spreadsheet first appeared in Forth Dimensions Volume 7, Issue 1 and 2. These issues contain the documentation for this disk.

703 - AUTOMATIC STRUCTURE CHARTS V.3 $40/43/45 ______
by Kim R. Harris
These tools for the analysis of large Forth programs were first presented at the 1985 FORML conference. Program documentation is contained in the 1985 FORML Proceedings.

Please specify disk size when ordering . . . . ________

FORTH INTEREST GROUP

P.O. BOX 8231 SAN JOSE, CALIFORNIA 95155 408/277-0668

Name ____________________

Member Number ____________________

Company ____________________

Address ____________________

City ____________________

State/Prov. __________ ZIP __________

Country ____________________

Phone ____________________

OFFICE USE ONLY

By		Date		Type	
Shipped By		Date			
UPS Wt.		Date			
USPS Wt.		Amt.			
BO Date		By			
Wt.		Amt.			

ITEM #	TITLE	AUTHOR	QTY	UNIT PRICE	TOTAL
108	MEMBERSHIP ➡				SEE BELOW

SUBTOTAL	
10% MEMBER DISCOUNT	
SUBTOTAL	
CA. RESIDENTS SALES TAX	
HANDLING FEE	$2.00
MEMBERSHIP FEE ☐ NEW ☐ RENEWAL $30/37/43	
TOTAL	

☐ Check enclosed (payable to: **FORTH INTEREST GROUP**)

☐ VISA ☐ MASTERCARD

Card # ____________________

Expiration Date ____________________

Signature ____________________

($15.00 minimum on charge orders)

PAYMENT MUST ACCOMPANY ALL ORDERS

MAIL ORDERS
Send to:
Forth Interest Group
P.O. Box 8231
San Jose, CA 95155

PHONE ORDERS
Call 408/277-0668 to place credit card orders or for customer service. Hours: Monday-Friday, 9am-5pm PST.

PRICES
All orders must be prepaid. Prices are subject to change without notice. Credit card orders will be sent and billed at current prices. $15 minimum on charge orders. Checks must be in US$, drawn on a US Bank. A $10 charge will be added for returned checks.

POSTAGE & HANDLING
Prices include shipping. A $2.00 handling fee is required with all orders.

SHIPPING TIME
Books in stock are shipped within five days of receipt of the order. Please allow 4-6 weeks for out-of-stock books (delivery in most cases will be much sooner).

SALES TAX
Deliveries to Alameda, Contra Costa, San Mateo, Los Angeles, Santa Cruz and San Francisco Counties, add 6½%. Santa Clara County, add 7%; other California counties, add 6%.

Continued from page 18.

Listing Two

Screen # 0

```
( Drawer     EGA LMI PC/Forth version         jbb 15:13 04/20/86 )
( Last change:   Screen  002                  jbb 16:25 04/29/86 )

This file requires an EGA adapter and display.  You must load
PC/FORTH's EGAGRAPH.COM before loading PC/FORTH in order to
load this file, or you will CRASH!!!  I guarantee it!


              Copyright 1986, by J. Brooks Breeden
                All commercial rights reserved.
You may freely use this program for personal non-commercial use.

This program is written in and for the Forth-83 implementation
        PC/FORTH 3.1 by Laboratory Microsystems, Inc.
```

Screen # 1

```
( Utilities required for demo                 jbb 15:20 04/20/86 )
: FG   FOREGROUND ; ( shorthand notation for PC/FORTH's names )
: BG   BACKGROUND ;
: HUE ( color - )   CREATE , DOES> ( - n )   @ ;
7 HUE GRAY      15 HUE WHITE      1 HUE BLUE
4 HUE RED       2 HUE GREEN

2VARIABLE CP              ( holds coordinates of graphic cursor)
: AT ( x y - )    CP 2! ;          ( set graphic cursor "AT" CP)
: DRAW ( x y - )    2DUP CP 2@ LINE AT ; ( draw line, update CP)
: V+ ( x y x1 y1 - x2 y2)    >R ROT + SWAP R> + ; ( add vectors)

: ?   @ . ;
-->
```

Screen # 2

```
( Drawer routines                             jbb 16:25 04/29/86 )
: DRAWER    " DRAWER.SCR " ;              ( file-name to return to)
: POINTFILE " USAEGA.SCR " ;         ( file-name holding points )

0 CONSTANT POINTS                         ( screen 0 in pointfile )
380 66 2CONSTANT ORIGIN            ( upper left corner of drawing)

\ Note: the POINTSBLOCK defined in POINTED.SCR is not the same.
: POINTSBLOCK ( n -)    POINTS BLOCK + 64 + ;  ( step +1 lines)

: PT# ( n - x y )    2* 2* POINTSBLOCK 2@ ORIGIN V+ ;
: DRAW-PTS ( hi lo - )    DO  I PT# DRAW  LOOP ;
-->
The above code is essentially all there is to it.  The remainder
of the screens illustrate using the drawer routines.
```

Screen # 3

```
( Steelchart regions of rainfall              jbb 11:14 04/29/86 )
: DRAWUSA    0 PT# AT 128 0  DRAW-PTS ;  ( draws outline of USA)
: DRAWSTEEL ( draws lines separating regions of equal rainfall)
   130 PT# AT    134 130 DRAW-PTS
   134 PT# AT    140 134 DRAW-PTS
   140 PT# AT    145 140 DRAW-PTS
   145 PT# AT    153 145 DRAW-PTS
   153 PT# AT    165 153 DRAW-PTS
   165 PT# AT    173 165 DRAW-PTS ;

: USAFRAME          ( draws a double border around the map area)
   GRAY FG   320 42 AT 631 42 DRAW 631 196 DRAW 320 196 DRAW 320
   42 DRAW    WHITE FG 316 38 AT 635 38 DRAW 635 200 DRAW 316
   200 DRAW 316 38 DRAW ;
: FILLUSA    480 100 GREEN  GRAY FLOOD ;               ( paint USA)
-->
```

Screen # 4

```
( Print region #s                             jbb 14:49 04/20/86 )
: .REGION#S    41 12 GOTOXY  GRAY FG      ( prints # of regions)
   ."  Regions of approx. similar rainfall."  WHITE FG
   ASCII 1 130 88 ORIGIN V+ GEMIT
   ASCII 2 110 45 ORIGIN V+ GEMIT
   ASCII 3 144 36 ORIGIN V+ GEMIT
   ASCII 4 156  6 ORIGIN V+ GEMIT
   ASCII 5  68 14 ORIGIN V+ GEMIT
   ASCII 6  56 50 ORIGIN V+ GEMIT
   ASCII 7  25 23 ORIGIN V+ GEMIT  ;
-->

Note: GEMIT is PC/FORTH for emiting a character in graphics
      mode at x y coordinates rather than column/row locations.
```

Screen # 5

```
( Draw image from file block                  jbb 11:17 04/29/86 )
\ this sequence will open/close required files.
: DRAWMAP                                      ( to draw the USA...)
   USAFRAME                           ( draw frame around image area)
   CLOSE-SCR                         ( close current screen "drawer")
   EMPTY-BUFFERS                       ( clear out any blocks stored)
   POINTFILE OPEN-SCR            ( open the screen-file "pointfile")
   GREEN FG DRAWUSA   FILLUSA                   ( draw the stuff...)
   RED FG DRAWSTEEL .REGION#S
   CLOSE-SCR                                ( close the drawing-file)
   EMPTY-BUFFERS                      ( clear any junk from buffers )
   DRAWER OPEN-SCR                              ( re-open "drawer")
   2DROP  ;                                  ( drop leftover flags)
-->
```

COMBINE THE RAW POWER OF FORTH WITH THE CONVENIENCE OF CONVENTIONAL LANGUAGES

HS/FORTH

Why HS/FORTH? Not for speed alone, although it is twice as fast as other full memory Forths, with near assembly language performance when optimized. Not even because it gives MANY more functions per byte than any other Forth. Not because you can run all DOS commands plus COM and EXE programs from within HS/FORTH. Not because you can single step, trace, decompile & dissassemble. Not for the complete syntax checking 8086/8087/80186 assembler & optimizer. Nor for the fast 9 digit software floating point or lightning 18 digit 8087 math pack. Not for the half megabyte LINEAR address space for quick access arrays. Not for complete music, sound effects & graphics support. Nor the efficient string functions. Not for unrivaled disk flexibility — including traditional Forth screens (sectored or in files) or free format files, all with full screen editors. Not even because I/O is as easy, but far more powerful, than even Basic. Just redirect the character input and/or output stream anywhere — display, keyboard, printer or com port, file, or even a memory buffer. You could even transfer control of your entire computer to a terminal thousands of miles away with a simple >COM <COM pair. Even though a few of these reasons might be sufficient, the real reason is that we don't avoid the objections to Forth — WE ELIMINATE THEM!

Public domain products may be cheap; but your time isn't. Don't shortchange yourself. Use the best. Use it now!

HS/FORTH, complete system: $395. with "FORTH: A Text & Reference" by Kelly and Spies, Prentice-Hall and "The HS/FORTH Supplement" by Kelly and Callahan

Visa Mastercard

HARVARD SOFTWORKS
PO BOX 69
SPRINGBORO, OH 45066
(513) 748-0390

```
Screen # 6
( DEMO: Setup                               jbb 14:52 04/20/86 )
: HEADER   0 0 GOTOXY  ." microLARCH Drainage Module: " ;
: GIVEN ." Given the following information: " ;
: SETUP  640X350 VMODE             ( EGA high-res graphics mode)
   BLUE BG  CLS
   WHITE FG HEADER 0 14 639 14 LINE 0 233 639 233 LINE
   RED FG 0 15 639 15 LINE 0 232 639 232 LINE
   GRAY FG 0 16 639 16 LINE 0 231 639 231 LINE ;
-->

Screen # 7
( Dummy variables for demo                  jbb 11:18 04/29/86 )
2VARIABLE D              23400 0  D 2!
2VARIABLE SLOPE          340 0 SLOPE 2!
2VARIABLE AC             125 0 AC 2!
2VARIABLE C              45 0 C 2!
VARIABLE REGION           3 REGION !
VARIABLE YEAR            10 YEAR !

\ this is a dummy floating point F@ and formatted F. for DEMO)
: F?  2@ <#  # # ASCII . HOLD #S #> TYPE SPACE ;

: >ENTRY-W  0 17 GOTOXY ;     ( dummy window control for DEMO)
-->

Screen # 8
( Problem                                   jbb 15:21 04/20/86 )
: DEMO   SETUP  DRAWMAP  WHITE FG 0 3 GOTOXY  GIVEN  GRAY FG
   1 5 GOTOXY ." Distance of flow (ft.): " D  F?
   1 6 GOTOXY ." Slope ( % )            :   " SLOPE  F?
   1 7 GOTOXY ." Coefficient of runoff :   " C  F?
   1 8 GOTOXY ." Region (Steel Chart)  :   " REGION ?
   1 9 GOTOXY ." Design Storm (year)   :  " YEAR ?
  1 10 GOTOXY ." Area (acres)          :   " AC F?
   >ENTRY-W  WHITE FG
   ." What is the Volume of runoff (Q) in cfs? "  GRAY FG CR ;
```

Index to Advertisers

Bradley Forthware - 11
Bryte - 6
CLICK Software - 8
Dash, Find & Associates - 37
Forth, Inc. - 10
Forth Interest Group - 7, 14, 19-22, 40
Harvard Softworks - 24
Laboratory Microsystems - 30
MicroMotion - 29
Miller Microcomputer Services - 33
Mountain View Press - 28
New Micros - 37
Next Generation Systems - 27
Palo Alto Shipping Company - 4
Software Composers - 2
SOTA - 34
Talbot Microsystems - 16
Tools Group - 35
UBZ Software - 13

Synonyms and Macros, Part 4

Compiler Macros

Victor H. Yngve
Chicago, Illinois

The important thing about compiler macros is that they postpone the normal compile-time action of words to a later compile time, thus providing us with valuable programming options. Let me explain.

The macro definitions written

MACRO <name> ... **END-MACRO**

that were previously introduced[1] may be called *compile-and-CMOVE macros.* They can be used to increase readability where a colon definition cannot be used because of interference with the return stack, e.g., in accessing loop indices, and they can be used to increase execution speed in time-critical parts of the code while preserving the readability of colon definitions.

However, there is the possibility of further improvements in readability where compile-and-CMOVE macros cannot be used because they would contain unpaired compiler words like **IF**, **BEGIN** or **LOOP**, or the Forth implementation has absolute rather than relative branch addresses. These cases would have to be coded in the familiar manner by what we can call *COMPILE-[COMPILE] compiler macros*: Each nonimmediate word is preceded by **COMPILE**, each immediate word by **[COMPILE]** and **IMMEDIATE** is appended to the definition. Some examples are given on screen fifty-eight, which shows macros for **IF-NOT** and **WHILE-NOT** suggested by Ed Petsche.[2]

But COMPILE-[COMPILE] compiler macros can be confusing and error prone. One has to remember which words are immediate and which not in order to choose between **[COMPILE]** and **COMPILE**, and it is easy to forget to append **IMMEDIATE**. Also, the resulting definitions are not very readable, thus tending to defeat an important reason for their use.

COMPILE-[COMPILE] compiler macros are even more difficult and awkward to use if the definition contains numbers. These were not covered in Jeffrey Soreff's article[3], which warrants close study, and it may not be immediately clear how to handle them. The use of numbers in COMPILE-[COMPILE] compiler macros is shown on the same screen in a suggested macro **IF#** for testing whether the number on the stack represents the ASCII code for a decimal digit.

The macro facility introduced here can take care of these cases by the words << (postpone) and >> (end postpone). Simply write

: <name> ... << ... >> ... ;

with the words of the macro enclosed between the angle brackets. It is not necessary to append **IMMEDIATE** to the definition. A Forth-83 style glossary entry is provided on screen sixty-two. These << ... >> macros are called *angle-bracket compiler macros.* They do nothing more than can be done with COMPILE-[COMPILE] compiler macros, namely they make a compiler word that postpones the immediate or nonimmediate action of the enclosed words, but they do it automatically. Their use is illustrated on screen sixty-three in the recoding of **IF-NOT**, **WHILE-NOT** and **IF#** as angle-bracket compiler macros. The increased readability due to removing the **COMPILE** and **[COMPILE]** clutter is especially apparent in the case of longer definitions like **IF#**.

Angle-bracket compiler macros can be nested in normal fashion together with colon definitions and compile-and-CMOVE macros. However, one cannot nest

[... **]**

inside an angle-bracket compiler macro to postpone the run-time action of the enclosed words. Use

>> ... <<

instead, thus dividing the compiler macro into two and compiling the included words normally. Note that

: <name> ... << ... >> ;

is a postponing version of

: <name> **[** ... **]** ... ;

The main purpose of using compiler macros is to postpone the execution of immediate words when this is necessary for resolving branch addresses or to postpone the processing of the input stream. Their main disadvantages for other purposes over compile-and-CMOVE macros are that they take up more space in the dictionary and they cannot be tested directly at the console without first embedding them in other words.

In a compiler macro, immediate words have their normal, immediate compile-time action postponed to when the compiler macro is used in compiling another word. With words like **."** this means compiling a string from the input stream for later output. With words like **IF** and **LOOP** this means compiling branches and resolving branch addresses. With **[']** this means compiling the compilation addresses of a word found in the input stream. For **LITERAL** it means compiling a number on the stack as a literal. In all these cases the normal run-time action resulting from the compile-time action is postponed to the time when the other word is executed. But for immediate words like **(** and **.(** nothing is compiled at the postponed compile time, for their normal compile-time action is to be executed.

Nonimmediate words such as **SWAP**, **+**, **CREATE**, **'** (tick) and **,** (comma) have their compilation postponed to the time when the compiler macro is used in compiling another word. Their run-time action is then postponed to the time when this other word is executed. At that time, **'** (tick) finds a Forth word in the input stream and leaves its compilation address on the stack, and words like **CREATE** make the following text in the input stream into the name field of a word being defined.

Thus there are three times associated with compiler macros where information is processed into a different form: a postpone time when each immediate word is compiled and each nonimmediate word is compiled with a preceding **COMPILE**, a compile time when the normal immediate or nonimmediate action of the words is taken in compiling another word, and a run time when the other word is executed. In this they are like **SYNONYM**, which has a time when the synonym is defined, a time when the synonym compiles the original word and a time when that word is executed.[4]

And in this they are different from compile-and-CMOVE macros, which have only a compile time and a run time, the run time being delayed by simply moving the compiled words into place with **CMOVE** without further processing. There are also *input-stream macros*, which have been explored by Don Taylor.[5] They also have only a compile time and a run time, but here the compile time is delayed by saving a copy of the source text in the dictionary.

In brief, the implementation of << and >> is straightforward and can be done entirely in Forth-83 with the help of the familiar nonstandard words defined in Forth-83 on screen fifty-nine. The word << (postpone) simply marks the word in which it appears as immediate, looks up each following word in the dictionary, determining whether it is immediate or not, and compiles it, preceding it by **COMPILE** if it is not immediate. This gives the same result as using a COMPILE-[COMPILE] compiler macro. If a word is not found, the auxiliary word **<NUMBER>** tries to convert it to a number and compile it appropriately as a literal with a following **[COMPILE] LITERAL**. Compilation is terminated by detecting >> (end postpone) in the input stream. The stretch of code between << and >> is marked to be used only while compiling.

For ease of learning and ease of use, it is important that numbers be treated the same as by the outer interpreter, which usually handles both double- and single-precision numbers. The standard does not dictate how double numbers should be represented in the input stream. The system implemented here in **<NUMBER>** is that if the number contains a period either initially, finally or in the middle, it is compiled as a double number. No record of the location of the decimal point is kept. Ideally **<NUMBER>** should be modified to use the double-

```
Screen # 58
( Examples of macros using COMPILE and [COMPILE] )

( from Ed Petsche, Forth Dimensions VII,3:6 )
: IF-NOT  COMPILE 0= [COMPILE] IF ; IMMEDIATE
: WHILE-NOT  COMPILE 0= [COMPILE] WHILE ; IMMEDIATE

: IF#  COMPILE DUP 47 [COMPILE] LITERAL COMPILE >
       COMPILE OVER 58 [COMPILE] LITERAL COMPILE <
       COMPILE AND [COMPILE] IF ; IMMEDIATE


Screen # 59
( Extensions needed to Forth-83 Required Word Set )
32 CONSTANT BL
: ASCII  ( -- n )  BL WORD 1+ C@
    STATE @ IF [COMPILE] LITERAL THEN ; IMMEDIATE
: ?COMP  STATE @ 0= ABORT" Compilation only " ;
: ?PAIRS - ABORT" Conditionals not paired " ;


Screen # 60
( Compiler Macros  <NUMBER>                          vhy 11/14/85 )

: <NUMBER> ( 0 0 addr -- ) ( Postpone Number Or Abort )
 DUP 1+ C@ ASCII - = DUP >R       ( neg flag to return stack )
   IF 1+ THEN                              ( 0  0  ad : nf )
 CONVERT DUP C@ ASCII . = DUP >R ( doub ) ( lo hi ad : df nf )
   IF CONVERT THEN C@ BL =       ( found? ) ( lo hi ff : df nf )
   IF  R> R> SWAP >R               ( neg? ) ( lo hi nf : df )
       IF  DNEGATE  THEN                   ( lo hi : df )
     SWAP [COMPILE] LITERAL COMPILE [COMPILE] LITERAL R>
       IF [COMPILE] LITERAL COMPILE [COMPILE] LITERAL
       ELSE DROP
       THEN
   ELSE  1 ABORT" Not found "
   THEN  ;



Screen # 61
( Compiler Macros  << ... >>                          vhy 11/14/85 )
: >>  ( -- ) 1 ABORT" Unpaired " ; IMMEDIATE ( End Postpone )
: <<  ( -- ) ( Postpone )
  ?COMP                         ( this word compile only )
  COMPILE ?COMP                 ( same for macro words )
  IMMEDIATE                     ( compiled word immediate )
  BEGIN
    BL WORD FIND                ( search dictionary )
    OVER ['] >> = NOT           ( not done? )
  WHILE  DUP                    ( was word found? )
    IF  1-                      ( was it nonimmediate? )
      IF COMPILE COMPILE THEN   ( for nonimmediate words only )
      ,                         ( postpone word )
    ELSE  0 ROT <NUMBER>        ( postpone number or abort )
    THEN
  REPEAT  DROP DROP  ;  IMMEDIATE

Screen # 62
( Compiler Macros  Glossary entry )

<<              --              I                      "postpone"
 A compiler word used in the form:
 : <name> ... << ... >> ... ;
 When << is used while compiling, it renders the definition
 immediate, compiles each immediate word between << and >> ,
 compiles each nonimmediate word with a preceding COMPILE,
 compiles each single number as a literal followed by
 [COMPILE] LITERAL , and each double number as two literals
 each followed by [COMPILE] LITERAL .  The stretch of words
 between << and >> are marked compile only.  The result is
 that the immediate or nonimmediate action of the words and
 numbers between << and >> is postponed by one step of compil-
 ation.  The word >> (end postpone) is freestanding and
 surrounded by spaces.
```

```
Screen # 63
( Compiler Macros -- Examples of usage )

: IF-NOT  << 0= IF >> ;
: WHILE-NOT  << 0= WHILE >> ;

: IF#  << DUP 47 > OVER 58 < AND IF >> ;

: FOR  << 1 SWAP DO >> ;
: NEXT  << -1 +LOOP >> ;

( Defines constants initialized to 0 that can be ticked )
: VALUE  CREATE 0 , DOES> @ ;

( TO also known as IS and ->   For usage see screen 65 )
: TO  ' >BODY  STATE @ IF << LITERAL ! >> ELSE ! THEN ;

Screen # 64
( Compiler Macros -- Examples of usage )

VARIABLE RECORD  138 RECORD !   ( postpone-time world record )
: IF>IS  << RECORD @ > IF >> ;
: IF>WAS  RECORD @ << LITERAL > IF >> ;
: IF>HAD-BEEN  [ RECORD @ ] LITERAL << LITERAL > IF >> ;

145 RECORD !                     ( compile-time world record )
: AWARDS?  DUP IF>IS ." Best.  Beats the current world record"
               ELSE DUP IF>WAS ." Better"
                        ELSE IF>HAD-BEEN ." Good"
                             ELSE ." Also ran"
             THEN THEN THEN  CR  ;

149 RECORD !                     ( run-time world record )
( now try n AWARDS? for different n )
```

Segmented Memory Model figures from page 13.

```
            +-------+-----+-----+     +-----+--------+
data        | nest  | cfa | cfa | ... | cfa | unnest |
segment     |       |     |     |     |     |        |
            +-------+-----+-----+     +-----+--------+

            ^
            |
            |
       compilation
         address
```

Figure One

```
          +-----------------+            +----------+---------+
code      |     addr of     |  code      | JMP nest | addr of |
segment   | executable code |  segment   |          |  body   |
          +-----------------+            +----------+---------+

          ^                              ^
          |                              |
          |                              |
    compilation                    compilation
      address                        address
```

Figure Two **Figure Three**

```
          +------------+---------------+------+
headers   | name field |  compilation  | link |
segment   |            |    address    |      |
          +------------+---------------+------+
```

Figure Four

NGS FORTH

A FAST FORTH, OPTIMIZED FOR THE IBM PERSONAL COMPUTER AND MS-DOS COMPATIBLES.

STANDARD FEATURES INCLUDE:

- 79 STANDARD
- DIRECT I/O ACCESS
- FULL ACCESS TO MS-DOS FILES AND FUNCTIONS
- ENVIRONMENT SAVE & LOAD
- MULTI-SEGMENTED FOR LARGE APPLICATIONS
- EXTENDED ADDRESSING
- MEMORY ALLOCATION CONFIGURABLE ON-LINE
- AUTO LOAD SCREEN BOOT
- LINE & SCREEN EDITORS
- DECOMPILER AND DEBUGGING AIDS
- 8088 ASSEMBLER
- GRAPHICS & SOUND
- NGS ENHANCEMENTS
- DETAILED MANUAL
- INEXPENSIVE UPGRADES
- NGS USER NEWSLETTER

A COMPLETE FORTH DEVELOPMENT SYSTEM.

PRICES START AT $70

NEW◆HP-150 & HP-110 VERSIONS AVAILABLE

NEXT GENERATION SYSTEMS
P.O.BOX 2987
SANTA CLARA, CA. 95055
(408) 241-5909

FORTH

The computer language for

increased...
EFFICIENCY

reduced.....
MEMORY

higher.......
SPEED

MVP-FORTH SOFTWARE

Stable...Transportable...
Public Domain...Tools

MVP-FORTH PROGRAMMER'S KIT

for IBM, Apple, CP/M, MS/DOS, Amiga, Macintosh and others. Specify computer.
$175

MVP-FORTH PADS,

a Professional Application Development System. Specify computer.
$500

MVP-FORTH EXPERT-2 SYSTEM

for learning and developing knowledge based programs.
$100

Word/Kalc,

a word processor and calculator system for IBM.
$150

Largest selection of FORTH books: manuals, source listings, software, development systems and expert systems.

Credit Card Order Number: 800-321-4103 (In California 800-468-4103)

Send for your FREE FORTH CATALOG

MOUNTAIN VIEW PRESS

PO BOX 4656
Mountain View, CA 94040

number scheme of the implementation in which it is installed.

Angle-bracket compiler macros have a variety of programming applications. For example, they are convenient for building new programming constructs. Their use in defining the looping words **FOR** and **NEXT** is illustrated on screen sixty-three. These words repeat a series of words n times by writing

n **FOR** ... **NEXT**

The index n is counted down to zero, and is available by using **I** inside the loop.

On the same screen, a high-level definition of the word **TO** (sometimes called -> or **IS**) is given using a compiler macro. This word works with special constants defined by **VALUE** (given on the same screen) which can have their value changed. One writes

VALUE SCORE

Then, executing

16 TO SCORE

changes the value of **SCORE** to sixteen. When the word **TO** is used, it obtains the address of the body of **SCORE** with '

```
Screen # 65
( An implementation of Eaker's CASE: Forth Dimensions II/3:37 )

VALUE OLD-DEPTH  ( Eaker used the fig-FORTH CSP for this. )
: CASE   ?COMP  OLD-DEPTH  DEPTH TO OLD-DEPTH  4 ; IMMEDIATE
: OF       4 ?PAIRS  << OVER = IF DROP >>      5 ;
: ENDOF    5 ?PAIRS  << ELSE >>                4 ;
: ENDCASE  4 ?PAIRS  << DROP >>
  BEGIN  DEPTH OLD-DEPTH = NOT WHILE  << THEN >>  REPEAT
  TO OLD-DEPTH  ;   ( restore depth-value for nesting )

: TEST   CASE  ( n -- )  ( illustrates usage of Eaker's CASE )
           10 OF ." TEN" ENDOF
           20 OF ." TWENTY" ENDOF
           30 OF ." THIRTY" ENDOF
                 ." NONE OF THE ABOVE"
         ENDCASE  CR  ;

Screen # 66
( Macro Sieve 1    PREFACE  VERSION E            vhy 12/5/85 )
8190 CONSTANT SIZE   CREATE FLAGS   SIZE ALLOT
MACRO    SET-FLAGS-TRUE                FLAGS SIZE 1 FILL END-MACRO
SYNONYM OCOUNT                         0
:        DO-FLAGS                   << SIZE 0 DO >>          ;
:        IF-TRUE                    << FLAGS I + C@ IF >>    ;
MACRO    GET-PRIME        ( - p )      I DUP + 3 +      END-MACRO
MACRO    FIRST-MULTIPLE ( p - p m )    DUP I +          END-MACRO
:        WHILE<SIZE     ( m - m ) << DUP SIZE < WHILE >>  ;
MACRO    SET-FALSE      ( m - m )      0 OVER FLAGS + C! END-MACRO
MACRO    NEXT-MULTIPLE ( p m - p n )   OVER +           END-MACRO
SYNONYM DROP-MULTIPLE                  DROP
SYNONYM PRINT-PRIME                    DROP
SYNONYM INC-COUNT                      1+
:        NEXT-FLAG                  << THEN LOOP >>          ;
:        PRINT-COUNT"               << . ." >>               ;

Screen # 67
( Macro Sieve 2   ALGORITHM  VERSION E           vhy 12/5/85 )

MACRO CANCEL-MULTIPLES ( prime -- prime )
   FIRST-MULTIPLE
   BEGIN  WHILE<SIZE SET-FALSE NEXT-MULTIPLE  REPEAT
   DROP-MULTIPLE  END-MACRO

: DO-PRIME
   SET-FLAGS-TRUE  OCOUNT
   DO-FLAGS
     IF-TRUE GET-PRIME CANCEL-MULTIPLES PRINT-PRIME INC-COUNT
   NEXT-FLAG
   PRINT-COUNT" Primes "  ;

: 10-TIMES  10 0 DO DO-PRIME LOOP  ;

```

>BODY and, if executing, uses it to store the number on the stack into the value. But if compiling, it instead compiles the address as a literal and then compiles **!** (store), thus postponing the actual change of the value to the run time of the word in which **TO SCORE** is contained. Thus a new value on the stack can be stored into **SCORE** with the speed of execution of

addr **!**

where addr is compiled as a literal.

An exercise in dealing separately with postpone time, compile time and run time is given on screen sixty-four. Suppose there is an event in which new world records are continually being made. The variable **RECORD** is updated at each of these times by storing in it the current new world record. If one then executes

n **AWARDS?**

at run time, the routine will successively test n against the different records stored at postpone time, at compile time and at run time, giving a printout with the wording dependent on whether n exceeds one or another of these.

The difference in behavior of compiled material and postponed material is further clarified in the compiler-macro implementation of Eaker's well-known **CASE** construct, which automatically compiles nested **IF** ... **ELSE** ... **THEN** constructs.[6] Here the compiler security apparatus for the **CASE** construct, involving testing pairs of numbers on the stack with **?PAIRS**, is executed at compile time, and the postponed material is compiled. The improved readability from earlier forms using COMPILE-[COMPILE] compiler macros is evident. It is of interest that Eaker's original implementation directly compiled **0BRANCH** and **BRANCH** and explicitly resolved the branch addresses, but this is not necessary, since the **IF**, **ELSE** and **THEN** can do this for us, as their immediate action is postponed to compile time.

Angle-bracket compiler macros would have only marginal utility if their only use was in implementing system words like **FOR**, **NEXT**, **TO** and **CASE** constructs, for these could be defined once and for all by a Forth system programmer as COMPILE-[COMPILE] macros. Their poor readability would be of little concern. But the genius of Forth is that everyone can be his own system programmer. The essence of Forth programming for anything but the simplest program is to invent an appropriate language tailored to the application, program the application in it and implement the special language in terms of Forth words in a prologue.

Thus the most important uses of angle-bracket compiler macros is in everyday programming to assist this process of organizing a program into appropriately named pieces. As an illustration, screens sixty-six and sixty-seven show the sieve benchmark program rewritten from version D in the last article[7], here using angle-bracket compiler macros as well as compile-and-CMOVE macros, synonyms and colon definitions. The further increase in readability is due to using angle-bracket compiler macros to name coherent stretches of code that happen to include unpaired conditional and looping words. If the implementation has absolute rather than relative addresses, the word **CANCEL-MULTIPLES** should be written as an angle-bracket macro as well. There is no increase in the running time of the benchmark, for exactly the same code is compiled as by the originally published sieve algorithm.

Let us conclude with a summary of when these various different types of definition should be used in the course of general Forth programming. Use

: <name> ... ;

for all normal purposes in Forth with the following exceptions:

Use

SYNONYM <name> <name>

for any definition with only one word in it. Synonyms run faster, take up less room in the dictionary and can be used for any word, whether it is immediate or not.

PORTABLE POWER WITH MasterFORTH

Whether you program on the **Macintosh**, the **IBM PC**, an **Apple II** series, a **CP/M** system, or the **Commodore 64**, your program will run unchanged on all the rest.

If you write for yourself, MasterFORTH will protect your investment. If you write for others, it will expand your marketplace.

Forth is interactive – you have immediate feedback as you program, every step of the way. Forth is fast, too, and you can use its built-in assembler to make it even faster. MasterFORTH's relocatable utilities and headerless code let you pack a lot more program into your memory. The resident debugger lets you decompile, breakpoint and trace your way through most programming problems. A string package, file interface and full screen editor are all standard features. And the optional target compiler lets you optimize your application for virtually any programming environment.

The package exactly matches *Mastering Forth* (Brady, 1984) and meets all provisions of the Forth-83 Standard.

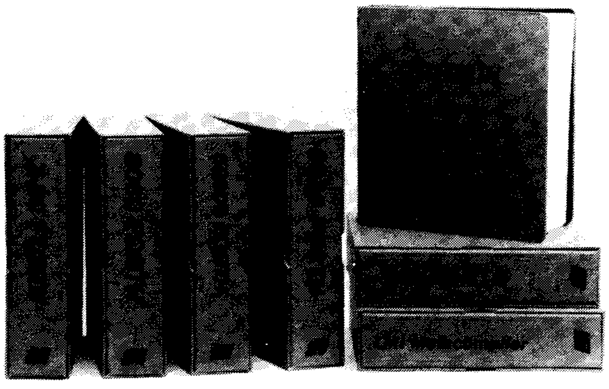
MasterFORTH standard package	$125
(Commodore 64 with graphics)	$100
Extensions	
Floating Point	$60
Graphics (selected systems)	$60
Module relocator (with utility sources)	$60
TAGS (Target Applic. Generation System) – MasterFORTH, target compiler and relocator	$495
Publications & Application Models	
Printed source listings (each)	$35
Forth-83 International Standard	$15
Model Library, Volumes 1-3 (each)	$40

(213) 821-4340

MICROMOTION

8726 S. Sepulveda Bl., #A171
Los Angeles, CA 90045

TOTAL CONTROL with *LMI FORTH*™



For Programming Professionals: an expanding family of compatible, high-performance, Forth-83 Standard compilers for microcomputers

For Development: Interactive Forth-83 Interpreter/Compilers
- 16-bit and 32-bit implementations
- Full screen editor and assembler
- Uses standard operating system files
- 400 page manual written in plain English
- Options include software floating point, arithmetic coprocessor support, symbolic debugger, native code compilers, and graphics support

For Applications: Forth-83 Metacompiler
- Unique table-driven multi-pass Forth compiler
- Compiles compact ROMable or disk-based applications
- Excellent error handling
- Produces headerless code, compiles from intermediate states, and performs conditional compilation
- Cross-compiles to 8080, Z-80, 8086, 68000, 6502, 8051, 8096, 1802, and 6303
- No license fee or royalty for compiled applications

For Speed: CForth Application Compiler
- Translates "high-level" Forth into in-line, optimized machine code
- Can generate ROMable code

Support Services for registered users:
- Technical Assistance Hotline
- Periodic newsletters and low-cost updates
- Bulletin Board System

Call or write for detailed product information and prices. Consulting and Educational Services available by special arrangement.

Laboratory Microsystems Incorporated
Post Office Box 10430, Marina del Rey, CA 90295
Phone credit card orders to: ***(213) 306-7412***

Overseas Distributors.
Germany: Forth-Systeme Angelika Flesch, Titisee-Neustadt, 7651-1665
UK: System Science Ltd., London, 01-248 0962
France: Micro-Sigma S.A.R.L., Paris, (1) 42.65.95.16
Japan: Southern Pacific Ltd., Yokohama, 045-314-9514
Australia: Wave-onic Associates, Wilson, W.A., (09) 451-2946

If : <name> ... ; cannot be used because the return will interfere with the return stack, or if it is desired to save the time of nesting and unnesting at run time, use

MACRO <name> ... **END-MACRO**

with the following exceptions:

If : <name> ... ; or **MACRO** <name> ... **END-MACRO** cannot be used because the code contains unmatched compiler words like **IF** or **DO**, or if **MACRO** <name> ... **END-MACRO** cannot be used because it contains branches compiled by words like **IF** or **LOOP** and the Forth implementation being used has absolute rather than relative branches, use

: <name> << ... >> ;

instead.
In addition, use

<< ... >>

in colon definitions along with other words to make compiler words that postpone compilation of the enclosed words for special purposes.

References

1. Yngve, Victor H. "Synonyms and Macros 2: Macros," *Forth Dimensions* VII/3.
2. Petsche, Ed. Letters to the Editor, *Forth Dimensions* VII/3.
3. Soreff, Jeffrey. "Macro Expansion in Forth," *Forth Dimensions* V/5.
4. Yngve, Victor H. "Synonyms and Macros 1: Synonyms," *Forth Dimensions* VII/3.
5. Taylor, Don. "Macro Generation," *Forth Dimensions* VII/1.
6. Eaker, Charles E. "Just in Case," *Forth Dimensions* II/3.
7. Yngve, Victor H. "Synonyms and Macros 3: Benchmark Readability," *Forth Dimensions* VII/4.

Shuffled Random Numbers

Leonard Zettel
Trenton, Michigan

In my opinion, the random number generator given by Leo Brodie in *Starting Forth* is indeed a very good one. While I have not checked it exhaustively, it seems to follow all the recommendations in Knuth[1], which is usually regarded as the definitive work on random number generation. It could therefore be regarded as best in its class, the class being sixteen-bit linear, congruential, pseudo-random number generators (if you really need to know what that means, read the reference).

That said, it must also be said that it shares a weakness shown by all generators of its type. To highlight the problem most dramatically, use **CHOOSE** to generate x,y pairs and plot them on your computer screen. When I do this on my Commodore 64 (using Forth extensions that come with SuperForth 64 by Parsec Research), with x going from 0 to 319 and y going from 0 to 159, what I get looks fine at first but eventually settles down to a tight corduroy pattern of diagonal lines. Roughly half the points never get plotted at all!

Now, if I ever see raindrops doing that on the pavement, I would get highly suspicious. Random behavior it ain't, though possibly good enough for many purposes to which random numbers are applied. When I used the numbers to study random walks, they turned out to be not good enough, because the walk kept insisting on avoiding large portions of the screen. In general, you will run into trouble if you use the numbers in bunches of n, with the first, second, etc. always used for the same purpose. In the plotting example, there are bunches of two, with the first number always used as the x coordinate and the second as the y. In general, you can expect the problem to get worse for larger n.

Fortunately, Forthwrights, there is a solution, which Knuth gives on page thirty-two. The answer is to shuffle the numbers. The screen shows a Forth implementation of his algorithm B. We put thirty-three random numbers generated using **RANDOM** into the array **RANDARRAY** using **RANDARRAY.INITIALIZE**. Then, when we need a number we invoke **SHRANDOM** to get a shuffled, random number. We use the thirty-third number in the **RANDARRAY** to calculate a position in the array. We then fetch to the stack the random number at that position, replacing it in the array with a new number from **RANDOM**. We then **DUP** the random draw we have on the stack and store one copy at the end of **RANDARRAY**, leaving the other on the stack as the result of **SHRANDOM**.

For good measure we have **SHOOSE**, which works just like Brodie's **CHOOSE** except that it uses **SHRANDOM** instead of **RANDOM**. For those of you tempted to tinker and, maybe, tempted to replace the multiply with a shift operation of some kind, note that **SHOOSE** and **CHOOSE** use the leftmost bits of the random number to generate the desired integer. This is recommended by Knuth, since the leftmost bits of the output of a congruential generator are, in many senses, "more random" than the right-hand bits.

Now for some comments and, if I can get away with it, some editorializing. I do not personally know of any way to get better quality random numbers (with a sixteen-bit modulus) than the screen presented here. On the other hand, being a somewhat lazy amateur, I have not tested the output nearly as thoroughly as I might have. In fact, the only tests I have run are the plots and the random walk. In addition, it is worth quoting Knuth (page 173), "Perhaps further research will show that even the random number generators recommended here are unsatisfactory; we hope this is not the case, but the history of the subject warns us to be cautious."

If Forth is indeed the best computer language in existence, and I firmly believe that it is, then it deserves to have only the very best routines put forward as part of its repertoire. You readers out there are the Forth community, and collectively you have more knowledge and experience than any single author. It would not surprise me at all if someone out there could improve on what I have presented. If you can, you have a duty to let the rest of us know about it; that is what communities are all about. If you don't want to work it up for publication, I would be glad to hear from you personally.

Reference

1. Knuth, Donald E. *The Art of Computer Programming*, Vol. 2, "Seminumerical Algorithms," 2nd ed., Addison-Wesley Publishing Co., Reading, Mass. 1981.

```
SCREEN #63
\ SHUFFLED RANDOM NUMBERS WITH STANDARD WORDS
VARIABLE RANDARRAY 64 ALLOT
: RANDARRAY.INITIALIZE ( ---)
 66 0 DO RANDOM RANDARRAY I + ! 2 +LOOP ;
RANDARRAY.INITIALIZE

: SHRANDOM ( --- U1)  \GET A SHUFFLED RANDOM NUMBER.
 RANDARRAY 64 + DUP @
 32 U* SWAP DROP 2*                   \RANDOM INDEX INTO ARRAY.
 RANDARRAY + DUP @ RANDOM ROT !        \DRAW & REPLACE RANDOM #.
 DUP ROT ! ;                          \NEW NUMBER AT THE END POS.

: SHOOSE ( U1 --- U2)
 RANDOMS U* SWAP DROP ;


```

The Multi-Dimensions of Forth

Glen B. Haydon
La Honda, California

Forth intrigued me when I first came across it six years ago. I had about ten years experience programming in a variety of high-level languages and some exposure to assembly languages. It always seemed that I needed to do things that were not immediately possible with the tools available in those languages. Furthermore, many of the languages required a mainframe computer for the compiler alone.

An acquaintance I met in a computer store dragged me along to a fourth Saturday meeting of the Forth Interest Group. I caught the disease. Many others have caught the disease. The symptoms vary, but its contagious nature is there.

I have tried to understand just what it is about Forth that is so contagious. I have found that Forth has many dimensions. Forth is a religion. Forth is a philosophy. Forth is a software kernel which emulates a hardware design.

Forth is a hardware processor. Forth is the assembly language for that processor. Forth is an operating system. Forth is a high-level language. Not all of these dimensions intersect.

Forth as a Religion

Franz Werfel in the introduction to the "Song of Bernadette" has the following quotation:

> "For those who believe in God, no explanation is necessary; for those who don't, no explanation is possible."

Those who believe in Forth have no problem with the requirements of reverse Polish notation, stack manipulations and the many unconventional ways of Forth. There is no way to explain Forth to experienced programmers who find the unusual characteristics of Forth impediments to "good" programming. A huge gap exists between the two points of view.

The newcomer to Forth is often better off if he has had no previous computer experience. After some time, he either becomes a member of the sect or he rejects it and goes on to something else. Those who end up using Forth let it become a part of their life.

Unfortunately, the religious image of Forth has been played up by the public media. It has had a way of turning off those who already have a religion.

Forth as a Philosophy

I find it difficult to pronounce "!". It is certainly not a common phonetic symbol. But "store" is a common Forth function. The symbol is like Chinese ideograms with no phonetic meaning.

I found that Forth functions are like the concepts associated with Chinese ideograms. I wrote a paper for the 1981 Rochester Forth Conference, "Forth and the Nature of Ideographic Thought." There is something very different between the way the Forth ideograms are assembled to convey an idea and more conventional ways of the classical high-level programming languages. The root of programming in Forth is to rethink the problem. Don't use thought processes derived from experience with the other programming languages.

After working closely with Charles Moore for about six months, I observed another dimension of Forth. Charles Moore has a way of immersing himself in the problem at hand. There was no such thing as an eight-hour work day. His entire life was immersed in the problem for days at a time.

Of course, his problems included the use of computers. Charles Moore attempted to understand all aspects of the problems. Many have heard him talk about the unreliability of hardware. He had to learn how to make the hardware, if possible. At the other end of programming computer applications, he found that most end users did not really understand the problem they were trying to solve. His approach was to immerse himself in all aspects. There were occasional escapes into science fiction and dreams of what things might be.

This total immersion in problem solving seems analogous to the ways of Zen. From time to time I have added pieces to a notebook which I have labeled, "Zen and the Art of Problem Solving." The title alone captures the spirit of the Forth philosophy.

Forth as a Software Kernel

The origins of Forth were based on trying to make existing hardware do the things that the applications required. It was a sort of virtual machine.

The Forth kernel simulates a small, necessary and sufficient set of functions which allows one to get on with the problems at hand. At the center of Forth implementations is some sort of kernel which allows the user to do what he needs to do. That Forth has been implemented on such a wide variety of processors is testimony to the virtue of a small, necessary and sufficient set of functions. It is a relatively simple matter to move the kernel from hardware system to hardware system.

A careful review of the necessary and sufficient functions shows that somewhere between sixty and seventy functions are convenient. It has been said that only eleven or twelve primitives programmed for a specific processor will provide all of the necessary operations with which to build a full, convenient set. But most implementations of Forth kernels include thirty or more specific functions unique to the specific hardware.

A unique design feature of Forth is its use of multiple stacks. Multiple stack architectures have been discussed since the mid-1950's. No true hardware implementation of a multiple-stack architecture was produced. Burroughs did provide an optimized set of pointer registers to utilize a part of main memory as an efficient stack. But the dedication of high-speed RAM to each stack apart from main memory is a relatively recent turn of things.

Forth as a Hardware Processor

There is a better way than emulating Forth in hardware: design the hardware to do the job. Many of the problems of implementing Forth on other processors just go away. Furthermore, a dramatic speed increase is possible — not just a factor of two or four, but a factor of one or two orders of magnitude.

Hartronics has produced a bit-slice design. Their product is proprietary but very efficient. In the last year or so, several other groups have addressed the problem of implementing a Forth kernel in hardware; not the least of these efforts has been that of Charles Moore. His group designed an architecture and committed it to silicon as the Novix 4000 chip. Several vendors are including that chip in systems. The systems are generally small and have low power requirements.

A group at Johns Hopkins have taken the basic design of the Novix 4000 device and expanded it to a thirty-two-bit processor on a chip. Again the functions are hard wired into the silicon.

A group in Hull, England, formed a company named Metaforth. They have produced a very fast board implementation which is becoming available. It has been optimized with custom components and is proprietary.

I have been associated with yet another design of hardware. The design features a writable control store and dedicated hardware stacks. The writable control store can be used to implement Forth or to experiment with other programming languages. It is able to run a functional Forth system as fast as those using a chip. The product is available today as a kit. All details are included in the wire-wrap kit to allow assembly from inexpensive, garden-variety components.

A writable control store machine is a true extension of the Forth philosophy of extensibility. It makes it possible to redefine and extend the actual processor functions. One should think of these machines as another processor design. The machine language for that processor can be made to execute a Forth kernel.

Forth as an Assembler

In conventional thinking, a Forth assembler provides access to the processor on which a software kernel is implemented. When the processor executes Forth functions, Forth becomes the assembler mnemonics for the processor. There simply is no other assembler.

With this level of maturity, the functions of the Forth kernel are directly compiled to the machine language of the Forth processor. As with many other processors, various fields in the machine language are directly coupled with the function of the hardware components.

Thus, one dimension of Forth is comparable with the assembly languages used with other processors.

Forth as an Operating System

Charles Moore found all of the overhead of an underlying operating system with its file structures and so on, completely unnecessary for his applications. He simply did not use the operating system's facilities in his work. Other early Forth users felt much the same way. They learned to work with their rudimentary operating system and found it most satisfactory. Forth was the operating system and the entire external storage was randomly accessed as a single file. What could be simpler?

But then there came those experienced with other operating systems and other programming languages. They expected to have available certain facilities. And many of them were using their hardware for other applications as well. They needed to have things "compatible."

The hardware emulation or simulation of a multiple stack system in Forth is not sufficient to yield a fully functional system.

When I started with the kernel from the *fig-FORTH Installation Manual*, everything seemed to work fine. I could load screens but I could not write to screens. Only if I were able to get something on a screen which was compatible with my drives, could I load it.

Interactively, I had to type in the necessary functions to allow text to be placed on a screen. Or, I could write the necessary functions in assembly language and have them available in my

FOR TRS-80 MODELS 1, 3, 4, 4P
IBM PC/XT, AT&T 6300, ETC.

WHICH ONE?

Which microcomputer word processor lets you create and edit without retyping, but *won't* slow down your creative process? Knows when to capitalize the first letter while replacing one phrase with another? Can outdent as well as indent? Will do typesetting at your command, even with proportional characters, right justification and tabbed columns? Lets you use the same (extra-capacity) data disks on IBM PC *and* TRS-80? And eases your learning with common-sense keystrokes, Help menus, good examples and a professionally authored manual?

Hint: it can integrate to communicate from home to office, and will interface with a database for form letters, data tables, and more!

It's the professional's word processor for your IBM PC, Compaq, or TRS-80 Model 1, 3 or 4:

FORTHWRITE
in
MMSFORTH

With an unusually powerful set of tools and an unusually easy way of helping you to use them.

The total software environment for IBM PC/XT, TRS-80 Model 1, 3, 4 and close friends.

- Personal License (required):

MMSFORTH V2.4 System Disk	$179.95

(TRS-80 Model 1 requires lowercase, DDEN, 1 40-track drive.)

- Personal License (additional modules):

FORTHCOM communications module	$ 49.95
UTILITIES	49.95
GAMES	39.95
EXPERT-2 expert system	69.95
DATAHANDLER	59.95
DATAHANDLER-PLUS (PC only, 128K req.)	99.95
FORTHWRITE word processor	99.95

- Corporate Site License Extensions . . . . from $1,000
- Bulk Distribution . . . from $500/50 units.
- Some recommended Forth books:

FORTH: A TEXT & REF. (best text!)	$ 18.95
THINKING FORTH (best on technique)	16.95
STARTING FORTH (popular text)	19.95

Shipping/handling & tax extra. No returns on software.
Ask your dealer to show you the world of MMSFORTH, or request our free brochure.

MILLER MICROCOMPUTER SERVICES
61 Lake Shore Road, Natick, MA 01760
(617) 653-6136

FORTH

FREEDOM OF CHOICE

SOTA Computing Systems Limited lets you choose between either the versatile figFORTH model or the popular 79 Standard. Each version is available for a number of popular computer systems including the IBM PC, XT and AT (or compatibles); the TRS-80 Model I, III and 4/4P; or any computer system running CP/M (version 2.x) or CP/M Plus (version 3.x). What's more, SOTA doesn't require you to enter into any awkward or expensive royalty or licensing arrangements. As long as your applications programs do not offer the end user access to the basic FORTH system, you are free to make as many copies of the compiled FORTH system as you please and distribute them as you wish. FORTH from SOTA is the FORTH of choice for both the novice and experienced programmer. Make it your choice now! Order your copy today.

79 STANDARD

fig MODEL

IBM®

TRS-80

CP/M

When you order from SOTA, both the fig model and 79 standard come complete with the following extra features at no additional charge:

• full featured string handling • assembler • screen editor • floating point • double word extension set • relocating loader • beginner's tutorial • comprehensive programmer's guide • exhaustive reference manual • unparalleled • technical support • source listings • • unbeatable price •

ORDER FORM

GENTLEMEN: Rush me my order!

☐ Enclosed is my: ☐ check ☐ money-order

☐ Please bill my: ☐ VISA ☐ MasterCard

for $89.95

TOTAL US funds

Please send me: ☐ 79 Standard FORTH ☐ figFORTH model

for the:

☐ IBM PC ☐ XT ☐ AT (and compatibles)

☐ TRS-80 Model I ☐ Model III ☐ Model 4 ☐ Model 4P

☐ CP/M Version 2.x ☐ CP/M Plus (Version 3.x)

For CP/M versions please note: 5 1/4" formats only and please specify computer type:

NAME: ____________

STREET: ____________

CITY/TOWN: ____________

STATE: ____________ ZIP: ____________

CARD TYPE: ____________ EXPIRY: ____________

CARD NO: ____________

SIGNATURE: ____________

ORDER TODAY 213-1080 Broughton Street Vancouver, British Columbia Canada • V6G 2A8

Order by Mail or Phone

MasterCard • (604) 688-5009 • VISA

State of the Art since 1981

SOTA Computing Systems Limited

IBM, TRS-80 and CP/M are registered trademarks of International Business Machine Corporation, Radio Shack and Digital Research respectively

assembled kernel. The problem was left to the user. With such a system, how could one imagine someone new to Forth being able to do much with it?

It is, perhaps, a fine distinction: is the editor a part of the operating system, or is it a higher-level language in its own right? In either case, the user can use whatever editor he likes. There are as many different ideas of an optimal system as there are people. And so with the extensibility of Forth, each implementor added the features he expected to find in his system. No two systems seemed to be the same. The file structures of UNIX, CP/M, MS-DOS, etc., are each different.

There simply is no standard in the computer industry. There are the ideas from the mainframe programmers and from the small, dedicated processor programmers. The extremes do not meet easily.

Many examples of the problems of dealing with foreign operating systems exist. The Apple Macintosh is perhaps the best example. Apple makes it almost impossible to use without their operating system. They do not want to give it up to a program.

Other operating system problems have to do with I/O devices. Not only do keyboards and monitors work differently from one machine to another, but the storage media varies. Even if you should be fortunate enough to have a common disk size, the formats of the disks vary!

Ideally, when Forth takes over a system, it also takes over all of the necessary drivers to use that system. The unit of external storage is a 1024K block. It is a problem of the implementation to translate actual disk sector sizes to logical 1024K blocks.

So, another dimension of Forth is that of being an operating system. There are, however, some Forth users who don't want to take advantage of this dimension. They forget that when they have this power, they can emulate or interface with any other operating system as the occasion demands. There is no reason why one cannot access data in files created by any other operating system.

Forth as a High-Level Language

Before I caught the Forth disease, I had a moderate amount of experience with a variety of high-level languages on mainframe computers. I found that PL/I provided me the most access to the functions I needed. But the PL/I compiler was large! There was no way it could be reduced to the 12K system I started with. Gradually, the small systems got larger, but a full implementation of PL/I would not even fit on many small machines today. Furthermore, even PL/I would not do everything I thought I wanted to do.

As a high-level language, the fig-FORTH Model left much to be desired. Many of the functions I expected just were not there. String functions were missing. There was no floating-point package. But I found that a fig-FORTH Model with a few custom bells and whistles served me well.

Then came a change of functional definitions for forty of the words in the fig-FORTH Model (suggested by the Standards Team). I really did not see much advantage to many of the changes. There was one major one in my book — the change of the **CREATE ... DOES>** function.

Reluctantly, I went ahead and made the forty changes to conform. This had the advantage that there were soon two tutorials — Leo Brodie's *Starting Forth* and Alan Winfield's *The Complete Forth*. The revision of the fig-FORTH Model with the forty functional changes became MVP-FORTH. I summarized my learning notes in a glossary that I am glad others have found useful.

At that point, I had a fully functional Forth which allowed me to go on with application programming. MVP-FORTH has remained stable for over four years. A substantial library of Forth functions is now available which run on the model I use. I will not be confined by the additional stipulation imposed by the 79 Standard. MVP-FORTH is not 79 Standard. I have no time to make the changes suggested by the 83 Standard and will not be confined by the additional stipulations.

My experience is not unique. Many other vendors have developed excellent implementations of a functional Forth. Some of them have kept their kernel and parts of the rest of their systems proprietary. This is fine. They have provided excellent documentation for their products This is a big help. However, most other vendors have made regular revisions. Some of the books on Forth for some vendor's dialects are no longer correct.

Now there is little point in changing your functional Forth just to be changing. Of course, there are many implementation techniques which could be adopted. One can use direct-threaded code or indirect-threaded code or token-threaded code. They make little difference in the application of Forth to specific projects There is little point quibbling about the details. There are better things to do.

After learning Forth, one should adopt a kernel and add only the necessary and sufficient routines to provide a uniquely functional Forth. For example, add your favorite editor and, only if you need it, add the floating-point functions. Collect your own library of Forth "goodies." The ability to do this is what is attractive about Forth.

As a high-level language, Forth is as extensible as you wish to make it. However, you need not be burdened with unnecessary and unused extensions.

Conclusions

I have taken you on a short tour of my attempts to understand Forth and where it is going.

Whatever you may want to call the process of immersing yourself in all aspects of a problem — a religion, a philosophy, even Zen — Forth embodies an approach to problem solving.

The essence of the Forth language is its kernel — an emulation of a hardware machine. As such, it provides a new approach to processor design. The new hardware machines with multiple stacks will open new doors to the computing world They will provide new alternatives to the conventional processors and the relatively new RISC machines.

If the primitive nature of a simple Forth operating system is satisfactory for your application, there is no reason to add the complexities of other systems. If not, you are free to add whatever operating system you wish.

Forth as a high-level language will shake down and take its place along with everything from BASIC to ADA. As has been demonstrated, one can even implement other high-level languages on a Forth software or hardware kernel. PROLOG has been implemented in a LISP which was implemented in Forth. I don't know what to call that combination.

The dimensions of Forth encompass many different aspects for any particular user. It is a little like Humpty-Dumpty in Alice in Wonderland. You need to make clear which combination of dimensions you are talking about, and to understand which others are talking about.

It is the richness of these many dimensions that makes Forth what it is.

THE TOOLS GROUP

66230 Forth Street
Desert Hot Springs, CA 92240
619/329-4625

Do you use Forth professionally?

Do you ever wish that you were working in an organization large enough to have a full time software tools group?

The Tools Group subscription support service is available to any Forth programmer. Tools are available for the Z80 and MC680x0 environments, running either CP/M 2.2, TPM III, or GEMDOS. iAPX 8x/28x and NS32x32 versions will be ready soon.

Access to the Tools Group programmers and library is through a Hartronix multiuser bulletin board system. Soon, the service will be available world wide through an X.25 based public data network at very low cost.

The Tools Group library is extensive, including tools like a 64 bit IEEE floating point package, transportable code between different operating systems, ASCII file support, automatic library manager, and much more. If the tool you need is not in the library, we will work with you to develop it.

Our motives are simple. We enjoy building real tools for real programmers. We want to help you. For a small annual fee, you may freely use our tools without royalties.

You don't need to tell anyone that you have a tools group -- you can just let them think you are Superprogrammer.

Stack Numbers by Name

Melvin Rosenfeld
Santa Barbara, California

When a word involves more than two or three numbers on the stack, the logistics of accessing them is often tedious. Sometimes the natural idea of putting the numbers in variables is not suitable. An example of this will be given shortly (example two).

I would like to offer a solution that I've devised. It consists of two words: { (the left brace) and ;; (a double semi-colon). Below are two illustrations of how these words are used.

Example One A definition of the function

$$f(x,y,z) = (x^2 + y)(xz + y)$$

is shown in Figure One. In that figure, the three words X, Y and Z that appear between the braces can be arbitrary. They are dummy names for the relevant stack entries. They will be in the dictionary only during the compilation of the word **F**. The stack effect of **F** will be (X Y Z -- answer).

Example Two As a consequence of the method of implementing the use of names for numbers on the stack, the function being defined can be used recursively. In this example, a word which solves the "Towers of Hanoi" puzzle will be given. I won't bother to remind readers what the puzzle is, since the point to be made is how neatly a recursive word can be defined. In this word, it is presumed that a word **OUTPUT** has already been defined (see Figure Two). **OUTPUT** takes two numbers a and b off the stack and reports, say by printing on the screen, "Take the top disk off peg a and place it on peg b."

In this example, one cannot use a variable to hold "the" value of, say, **#DISKS** since many such values must be kept while the recursive function **HANOI** wends its way down.

Discussion of the Method

Since { and ;; are used to define a word, there are two aspects to consider: first, the run-time effect of their use on the word being defined, and second, their operation during compilation of the word. Let us consider the word **F** from example one. Prior to its compilation, it is necessary to have the following words in the dictionary:

```
0 VARIABLE STACKTOP
: MARKSTACK  SP@  STACKTOP  ! ;
: C1  STACKTOP  @  0  +  @ ;
: C2  STACKTOP  @  2  +  @ ;
: C3  STACKTOP  @  4  +  @ ;
```

The dictionary entry for **F** (after it has been compiled) will, in its parameter field, contain the sequence of CFA's of the words comprising its definition. The names of this sequence of words are, in order,

```
MARKSTACK  C3  DUP  *  C2
+  C3  C1  *  C2 +  *
```

then a literal, in this case three (3), followed by the CFA of another word, **SHIFTSTACK** and finally **;S**.

The word **SHIFTSTACK** must also be in the dictionary before **F** can be compiled. Its name suggests what it does, but its definition is best left until later. Since the temporary name **X** for the third number on the stack has been replaced by **C3**, there is no reason for **X** to be in the dictionary.

Next we consider the compilation phase of the word **F**. Consider the string of symbols

```
: F { X Y Z } ... ;;
```

When the interpreter reaches : it of course begins to compile the next word, namely **F**. It is thus in compile mode when { is encountered. { is an immediate word and, instead of being compiled, does the following:

1. Concludes the definition of **F** (so that, for the moment, it is a no-op).
2. Counts the number of words in the input stream until } (three words, in this case).
3. Defines the first word, namely **X**, to be an immediate word. The action of **X** will be to place the CFA of **C3** into the next place in the dictionary. It similarly defines **Y** and **Z**.
4. Begins compilation of the next word in the input stream, namely }. The interpreter continues to compile the words in the input stream until ;; is encountered. This latter word is also immediate. It does the following:

1. Compiles the literal 3 and one last word, namely **SHIFTSTACK** into the definition of }.
2. Concludes the definition of }.
3. Shifts within the dictionary the parameter field of } so that it abuts the parameter field of **F** with the effect that the definition of } becomes the definition of **F**.
4. Erases from the dictionary the words **X**, **Y**, **Z** and }.

There is one additional word that may be useful, namely **;;S**. It acts very much like ;; but doesn't cause the stack to be shifted at the conclusion of the run-time action of **F**.

Time Penalty for Naming Numbers

The time spent in compiling seems unimportant, since it is undoubtedly made up for in time saved during programming. The run-time cost is important (if the word will be used many times in a program). Figure Three shows a short test to check the overhead of using this technique.

The overhead — the difference between **rightway** and **wrongway** — is 43/20000 seconds per implementation, about 1/465. Most of the time is taken up by the word **SHIFTSTACK**. Figure Four provides a second test, in which **;;S** was used and, therefore, **SHIFTSTACK** isn't needed. The overhead per implementation without **SHIFTSTACK** is 1/1176 seconds.

Since only a few words (:, **MARKSTACK**, **SHIFTSTACK**, **C1**, **C2**, **C3** ... **C10**) are used during run time, if they were written in machine language, I suspect **test2** would hardly be slower than **test1** and there would be little penalty for using what seems to me a very natural way of accessing the stack.

The version of Forth that I've used is Wycove Forth for the TI 99/4. I believe it is a variant of fig-FORTH. I have used the word **move** which in this Forth seems to diverge from the standard.

In Wycove Forth:

addr1 addr2 count **MOVE**

Moves the block of main memory bytes from addr1 to addr2. The length of the block in bytes is given as count. The direction of the move is determined so that no overlap occurs, with the word **CMOVE** or **<CMOVE** called to actually move the bytes. Data is moved one byte at a time.

In fig-FORTH:

addr1 addr2 n **MOVE**

Moves n sixteen-bit cells, beginning at addr1, to the memory locations beginning at addr2. The move proceeds from low to high memory. (From *Forth Fundamentals* Vol. 2, by Kevin McCabe.)

```
: F ( X Y Z ) X DUP * Y + X Z * Y + * ;;
```

Figure One

```
: HANOI ( PEG1 PEG2 PEG3 *DISKS )
        *DISKS 1 =
           IF PEG1 PEG2 OUTPUT
           ELSE PEG1 PEG3 PEG2 *DISKS 1- HANOI
                PEG1 PEG2 OUTPUT
                PEG3 PEG2 PEG1 *DISKS 1- HANOI
           THEN ;;
```

Figure Two

```
: rightway * DROP ;
: wrongway ( x y ) x y * DROP ;;
: test1 20000 0 DO I I rightway LOOP ;
: test2 20000 0 DO I I wrongway LOOP ;

The time for test1  was 7.2 seconds
The time for test2 was 50.2 seconds.
```

Figure Three

```
: RIGHTWAY   OVER OVER * DROP ;
: WRONGWAY   ( A B )   A B * DROP ;;S
: TEST3   20000 0 DO RIGHTWAY LOOP ;
: TEST4   20000 0 DO WRONGWAY LOOP ;

The time for TEST3 was 7.3 seconds.
The time for TEST4 was 24.3 seconds.
```

Figure Four

DASH, FIND & ASSOCIATES

Our company, DASH, FIND & ASSOCIATES, is in the business of placing FORTH Programmers in positions suited to their capabilities. We deal only with FORTH Programmers and companies using FORTH. If you would like to have your resumé included in our data base, or if you are looking for a FORTH Programmer, contact us or send your resumé to:

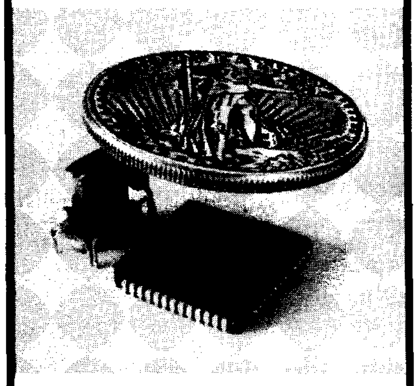
DASH, FIND & ASSOCIATES
808 Dalworth, Suite B
Grand Prairie TX 75050
(214) 642-5495

DFND

Committed to Excellence

All the parts needed to make the

SMALLEST PROGRAMMABLE FORTH SYSTEM:



&
+5V (9 mA, typical @ 2 MHz)
TTL Serial In
TTL Serial Out
Ground

$50 covers price of parts and manual in singles, $20 covers cost of chip alone in 10,000 quantity. $20 gold piece (not included) shown covering chip to illustrate actual size.

The F68HC11 features: 2 Serial Channels, 5 Ports, 8 Channel 8-bit A/D, major timer counter subsystem, Pulse Accumulator, Watchdog Timer, Computer Operating Properly (COP) Monitor, 512 bytes EEPROM, 256 bytes RAM, 8K byte ROM with FORTH-83 Standard implementation.

Availability: F68HC11 Production units with Max-FORTH™ in internal ROM available 4Q/86. Volume quantity available 1Q/87. X68HC11 emulator with Max-FORTH™ in external ROM available now. NMIX-0022 68HC11 Development System boards available now: $290.00.

New Micros, Inc.
808 Dalworth
Grand Prairie, TX 75050
(214) 642-5494

NEW MICROS INC.
808 DALWORTH
GRAND PRAIRIE, TEXAS 75050
214/642-5494

FIG Chapters

U.S.

• ALABAMA
Huntsville FIG Chapter
Call Tom Konantz
205/881-6483

• ALASKA
Kodiak Area Chapter
Call Horace Simmons
907/486-5049

• ARIZONA
Phoenix Chapter
Call Dennis L. Wilson
602/956-7678

Tucson Chapter
Twice Monthly,
2nd & 4th Sun., 2 p.m.
Flexible Hybrid Systems
2030 E. Broadway #206
Call John C. Mead
602/323-9763

• ARKANSAS
Central Arkansas Chapter
Twice Monthly, 2nd Sat., 2p.m. &
4th Wed., 7 p.m.
Call Gary Smith
501/227-7817

• CALIFORNIA
Los Angeles Chapter
Monthly, 4th Sat., 10 a.m.
Hawthorne Public Library
12700 S. Grevillea Ave.
Call Phillip Wasson
213/649-1428

Monterey/Salinas Chapter
Call Bud Devins
408/633-3253

Orange County Chapter
Monthly, 4th Wed., 7 p.m.
Fullerton Savings
Talbert & Brookhurst

Fountain Valley
Monthly, 1st Wed., 7 p.m.
Mercury Savings
Beach Blvd. & Eddington
Huntington Beach
Call Noshir Jesung
714/842-3032

San Diego Chapter
Weekly, Thurs., 12 noon
Call Guy Kelly
619/268-3100 ext. 4784

Sacramento Chapter
Monthly, 4th Wed., 7 p.m.
1798-59th St., Room A
Call Tom Ghormley
916/444-7775

Bay Area Chapter
Silicon Valley Chapter
Monthly, 4th Sat.
FORML 10 a.m., Fig 1 p.m.
H-P Auditorium
Wolfe Rd. & Pruneridge,
Cupertino
Call John Hall 415/532-1115
or call the FIG Hotline:
408/277-0668

Stockton Chapter
Call Doug Dillon
209/931-2448

• COLORADO
Denver Chapter
Monthly, 1st Mon., 7 p.m.
Cliff King
303/693-3413

• CONNECTICUT
Central Connecticut Chapter
Call Charles Krajewski
203/344-9996

• FLORIDA
Orlando Chapter
Every two weeks, Wed., 8 p.m.
Call Herman B. Gibson
305/855-4790

Southeast Florida Chapter
Monthly, Thurs., p.m.
Coconut Grove area
Call John Forsberg
305/252-0108

Tampa Bay Chapter
Monthly, 1st. Wed., p.m.
Call Terry McNay
813/725-1245

• GEORGIA
Atlanta Chapter
3rd Tuesday each month, 6:30 p.m.
Computone Cottilion Road
Call Ron Skelton
404/393-8764

• ILLINOIS
Cache Forth Chapter
Call Clyde W. Phillips, Jr.
Oak Park
312/386-3147

Central Illinois Chapter
Urbana
Call Sidney Bowhill
217/333-4150

Fox Valley Chapter
Call Samuel J. Cook
312/879-3242

Rockwell Chicago Chapter
Call Gerard Kusiolek
312/885-8092

• INDIANA
Central Indiana Chapter
Monthly, 3rd Sat., 10 a.m.
Call John Oglesby
317/353-3929

Fort Wayne Chapter
Monthly, 2nd Tues., 7 p.m.
IPFW Campus
Rm. 138, Neff Hall
Call Blair MacDermid
219/749-2042

• IOWA
Iowa City Chapter
Monthly, 4th Tues.
Engineering Bldg., Rm. 2128
University of Iowa
Call Robert Benedict
319/337-7853

Central Iowa FIG Chapter
Call Rodrick A. Eldridge
515/294-5659

Fairfield FIG Chapter
Monthly, 4th day, 8:15 p.m.
Call Gurdy Leete
515/472-7077

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

• LOUISIANA
New Orleans Chapter
Call Darryl C. Olivier
504/899-8922

• MASSACHUSETTS
Boston Chapter
Monthly, 1st Wed.
Mitre Corp. Cafeteria
Bedford, MA
Call Bob Demrow
617/688-5661 after 7 p.m.

• MICHIGAN
Detroit Chapter
Monthly, 4th Wed.
Call Tom Chrapkiewicz
313/562-8506

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

• MISSOURI
Kansas City Chapter
Monthly, 4th Tues., 7 p.m.
Midwest Research Institute
MAG Conference Center
Call Linus Orth
913/236-9189

St. Louis Chapter
Monthly, 1st Tues., 7 p.m.
Thornhill Branch Library
Contact Robert Washam
91 Weis Dr.
Ellisville, MO 63011

• NEVADA
Southern Nevada Chapter
Call Gerald Hasty
702/452-3368

• NEW HAMPSHIRE
New Hampshire Chapter
Monthly, 1st Mon., 6 p.m.
Armtec Industries
Shepard Dr., Grenier Field
Manchester
Call M. Peschke
603/774-7762

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

• NEW YORK
FIG, New York
Monthly, 2nd Wed., 8 p.m.
Queens College
Call Ron Martinez
212/517-9429

Rochester Chapter
Bi-Monthly, 4th Sat., 2 p.m.
Hutchinson Hall
Univ. of Rochester
Call Thea Martin
716/235-0168

Rockland County Chapter
Call Elizabeth Gormley
Pearl River
914/735-8967

Syracuse Chapter
Monthly, 3rd Wed., 7 p.m.
Call Henry J. Fay
315/446-4600

• OHIO
Akron Chapter
Call Thomas Franks
216/336-3167

Athens Chapter
Call Isreal Urieli
614/594-3731

Cleveland Chapter
Call Gary Bergstrom
216/247-2492

Cincinatti Chapter
Call Douglas Bennett
513/831-0142

Dayton Chapter
Twice monthly, 2nd Tues., &
4th Wed., 6:30 p.m.
CFC 11 W. Monument Ave.
Suite 612

Dayton, OH
Call Gary M. Granger
513/849-1483

• **OKLAHOMA**

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

• **OREGON**

Greater Oregon Chapter
Monthly, 2nd Sat., 1 p.m.
Tektronix Industrial Park
Bldg. 50, Beaverton
Call Tom Almy
503/692-2811

• **PENNSYLVANIA**

Philadelphia Chapter
Monthly, 4th Sat., 10 a.m.
Drexel University, Stratton Hall
Call Melanie Hoag or Simon Edkins
215/895-2628

• **TENNESSEE**

East Tennessee Chapter
Monthly, 2nd Tue., 7:30 p.m.
Sci. Appl. Int'l. Corp., 8th Fl.
800 Oak Ridge Turnpike, Oak Ridge
Call Richard Secrist
615/483-7242

• **TEXAS**

Austin Chapter
Contact Matt Lawrence
P.O. Box 180409
Austin, TX 78718

Dallas/Ft. Worth Metroplex Chapter
Monthly, 4th Thurs., 7 p.m.
Call Chuck Durrett
214/245-1064

Houston Chapter
Call Dr. Joseph Baldwin
713/749-2120

Periman Basin Chapter
Call Carl Bryson
Odessa
915/337-8994

• **UTAH**

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

• **VERMONT**

Vermont Chapter
Monthly, 3rd Mon., 7:30 p.m.
Vergennes Union High School
Rm. 210, Monkton Rd.
Vergennes, VT
Call Don VanSyckel
802/388-6698

• **VIRGINIA**

First Forth of Hampton Roads
Call William Edmonds
804/898-4099

Potomac Chapter
Monthly, 2nd Tues., 7 p.m.
Lee Center
Lee Highway at Lexington St.
Arlington, VA
Call Joel Shprentz
703/860-9260

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

• **WISCONSIN**

Lake Superior FIG Chapter
Monthly, 2nd Fri., 7:30 p.m.
University of Wisconsin
Superior
Call Allen Anway
715/394-8360

Milwaukee Area Chapter
Call Donald H. Kimes
414/377-0708

MAD Apple Chapter
Contact Bill Horzon
129 S. Yellowstone
Madison, WI 53705

FOREIGN

• **AUSTRALIA**

Melbourne Chapter
Monthly, 1st Fri., 8 p.m.
Contact Lance Collins
65 Martin Road
Glen Iris, Victoria 3146
03/29-2600

Sydney Chapter
Monthly, 2nd Fri., 7 p.m.
John Goodsell Bldg.
Rm. LG19
Univ. of New South Wales
Sydney
Contact Peter Tregeagle
10 Binda Rd., Yowie Bay
02/524-7490

• **BELGIUM**

Belgium Chapter
Monthly, 4th Wed., 20:00h
Contact Luk Van Loock
Lariksdreff 20
2120 Schoten
03/658-6343

Southern Belgium FIG Chapter
Contact Jean-Marc Bertinchamps
Rue N. Monnom, 2
B-6290 Nalinnes
Belgium
071/213858

• **CANADA**

Alberta Chapter
Call Tony Van Muyden
403/962-2203

Nova Scotia Chapter
Contact Howard Harawitz
227 Ridge Valley Rd.
Halifax, Nova Scotia B3P2E5
902/477-3665

Southern Ontario Chapter
Quarterly, 1st Sat., 2 p.m.
General Sciences Bldg., Rm. 312
McMaster University
Contact Dr. N. Solntseff
Unit for Computer Science
McMaster University
Hamilton, Ontario L8S4K1
416/525-9140 ext. 3443

Toronto FIG Chapter
Contact John Clark Smith
P.O. Box 230, Station H
Toronto, ON M4C5J2

• **COLOMBIA**

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

• **ENGLAND**

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

• **FRANCE**

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

• **GERMANY**

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

• **HOLLAND**

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

FIG des Alpes Chapter
Contact: Georges Seibel
19 Rue des Hirondelles
74000Annely
50 57 0280

• **IRELAND**

Irish Chapter
Contact Hugh Doggs
Newton School
Waterford
051/75757 or 051/74124

• **ITALY**

FIG Italia
Contact Marco Tausel
Via Gerolamo Forni 48
20161 Milano
02/645-8688

• **JAPAN**

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

• **NORWAY**

Bergen Chapter
Kjell Birger Faeraas
Hallskaret 28
Ulset
+47-5-187784

• **REPUBLIC OF CHINA**

R.O.C.
Contact Ching-Tang Tzeng
P.O. Box 28
Lung-Tan, Taiwan 325

• **SWEDEN**

Swedish Chapter
Hans Lindstrom
Gothenburg
+46-31-166794

• **SWITZERLAND**

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

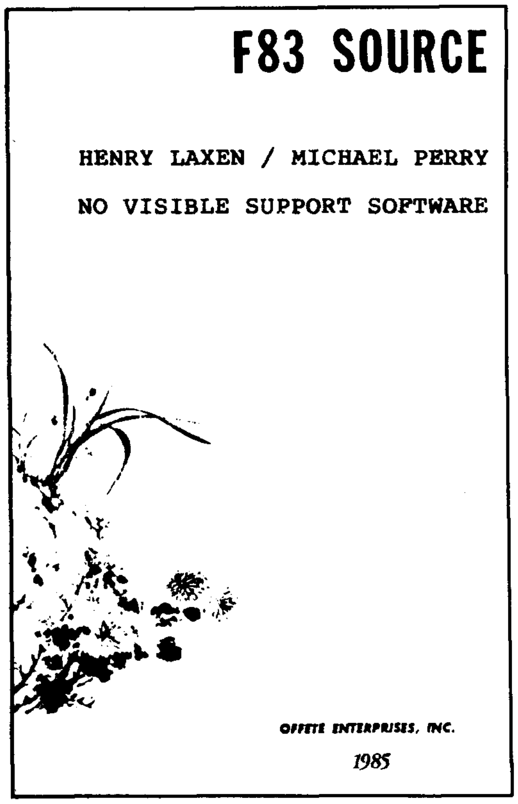
SPECIAL GROUPS

Apple Corps Forth Users Chapter
Twice Monthly, 1st &
3rd Tues., 7:30 p.m.
1515 Sloat Boulevard, #2
San Francisco, CA
Call Robert Dudley Ackerman
415/626-6295

Baton Rouge Atari Chapter
Call Chris Zielewski
504/292-1910

FIGGRAPH
Call Howard Pearlmutter
408/425-8700

NOW AVAILABLE



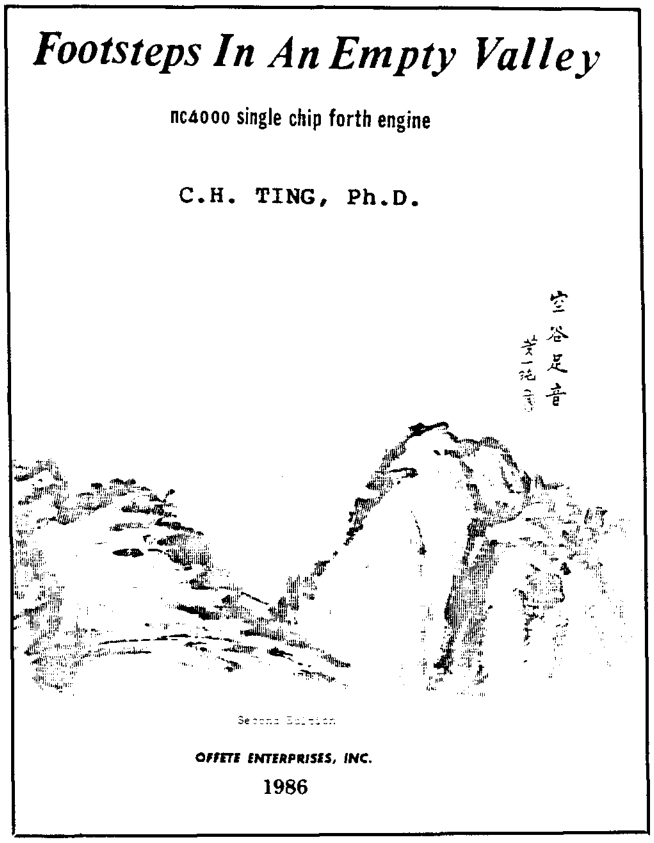
F83 SOURCE

HENRY LAXEN / MICHAEL PERRY

NO VISIBLE SUPPORT SOFTWARE

OFFETE ENTERPRISES, INC.

1985



Footsteps In An Empty Valley

nc4000 single chip forth engine

C.H. TING, Ph.D.

空谷足音

黄一沱 [illegible]

Second Edition

OFFETE ENTERPRISES, INC.

1986

FORTH INTEREST GROUP fig

$25 EACH

FORTH INTEREST GROUP fig

FROM THE FORTH INTEREST GROUP

FORTH INTEREST GROUP
P. O. Box 8231
San Jose, CA 95155

BULK RATE
U.S. POSTAGE
PAID
Permit No. 3107
San Jose, CA

Address Correction Requested