INTRODUCING
THE PC-RISC SYSTEM
For High-Speed Concurrent Parallel Processing
ADD UP TO 30 MIPS TO YOUR PC!

THE PC-RISC SYSTEM

The PC-RISC System can turn your PC into a Forth "mini-supercomputer." This high speed parallel processing system can be configured using your PC, XT, or AT as the host by adding up to six PC4000s, an add-in card with the NC4000 Forth engine on board. PCX operating and development software is included. The PC-RISC System runs SCForth, cmForth, and Delta-C. PC4000/4MIPS $1295 each. PC4000/5MIPS $1495 each. OEM pricing available.

The PC-RISC System, Delta Board, and Delta Development System come fully assembled and tested with a 90-day warranty, communication software, user manual, a Forth development system, demo programs, and customer support using the Silicon Composers Bulletin Board (415) 322-2891.

SILICON COMPOSERS
210 California Avenue, Suite I
Palo Alto, CA 94306
(415) 322-8763

Formerly
SOFTWARE COMPOSERS
FORTH

FLEXIBLE TEST ENVIRONMENT • JOHN MULLEN
9
Forth's interactiveness simplifies testing and debugging, but some words or components can be difficult to verify. The author describes a test environment that optionally prints messages at run time, without modification of source screens. He describes the environment's implementation, and how it may be used.

FORGETTABLE INTERNAL NAMES • MICHAEL HORE
12
Internal names are only used locally in an application. Dewey Val Schorre presented three words (INTERNAL, EXTERNAL, and MODULE) to implement this notion. The code here will allow a whole module to be loaded temporarily and then dismissed from memory, leaving subsequent definitions intact.

CONSUMERIZED FORTH • KEN TAKARA
18
Many applications hide Forth completely, but a robotic construction kit needs to include a robot control language. What do you include in a language for computer novices? What are the selection criteria?

EXECUTION SECURITY • G.R. JAFFRAY, JR.
25
One reason Forth runs so fast is that it does not have the built-in checks of other high-level languages. During development, this means crashes that could be avoided with this patch that provides execution security.

1987 ROCHESTER FORTH CONFERENCE • JERRY SHIFRIN
26
Technical discussion at the University of Rochester this year included several talks on the massively parallel processors being programmed in Forth, an interesting panel discussion on standards, and many other subjects.

STARFLIGHT, STAR BRIGHT • INTERVIEW WITH TIM LEE
29
Two programmer-decades of work have put more alien races, planets, and realistic interaction onto two full disks than a user-century of play may uncover. This game is breaking sales records at warp speed.

CANDIDATES’ STATEMENTS • FIG BOARD NOMINEES
40
Several individuals have been nominated as candidates for the vacant positions on the Forth Interest Group's Board of Directors. They were asked to submit statements of their candidacy, which are printed here in full.

EDITORIAL
4
LETTERS
5
ADVERTISERS INDEX
28
FIG CHAPTERS
42
This year’s National Forth Convention will be the occasion to celebrate ten years of the Forth Interest Group. C.H. Ting, the program director, points out that by understanding where Forth has come from, we can better see where it is going. The convention program promises to deliver presentations from people who have figured prominently in the history of the Forth Interest Group, who have made significant contributions to Forth’s evolution, and who have witnessed the key turning points in our industry.

This will be a time to look both backward and forward in time, to assimilate the collective experiences of the past and to gather ourselves for the future. By understanding where Forth has gone, and to gather ourselves for the future. By understanding where Forth has gone, and by looking forward, we can better see where it is going.

This year’s National Forth Convention will be the occasion to celebrate ten years of the Forth Interest Group. C.H. Ting, the program director, points out that by understanding where Forth has come from, we can better see where it is going. The convention program promises to deliver presentations from people who have figured prominently in the history of the Forth Interest Group, who have made significant contributions to Forth’s evolution, and who have witnessed the key turning points in our industry.

This will be a time to look both backward and forward in time, to assimilate the collective experiences of the past and to gather ourselves for the future. By understanding where Forth has gone, and by looking forward, we can better see where it is going.

This year’s National Forth Convention will be the occasion to celebrate ten years of the Forth Interest Group. C.H. Ting, the program director, points out that by understanding where Forth has come from, we can better see where it is going. The convention program promises to deliver presentations from people who have figured prominently in the history of the Forth Interest Group, who have made significant contributions to Forth’s evolution, and who have witnessed the key turning points in our industry.

This will be a time to look both backward and forward in time, to assimilate the collective experiences of the past and to gather ourselves for the future. By understanding where Forth has gone, and by looking forward, we can better see where it is going.

This year’s National Forth Convention will be the occasion to celebrate ten years of the Forth Interest Group. C.H. Ting, the program director, points out that by understanding where Forth has come from, we can better see where it is going. The convention program promises to deliver presentations from people who have figured prominently in the history of the Forth Interest Group, who have made significant contributions to Forth’s evolution, and who have witnessed the key turning points in our industry.

This will be a time to look both backward and forward in time, to assimilate the collective experiences of the past and to gather ourselves for the future. By understanding where Forth has gone, and by looking forward, we can better see where it is going.
Sorting Out Batcher's

Dear Mr. Ouverson,

Many thanks for the article on "Batcher's Sort," by John Konopka (FD VIII/4). I am about the present this to my students in a Forth class here.

When typing BSORT, I found two offsetting errors. The program runs correctly, but gives a misleading impression about the role of QQ in BSORT. Consider the BSORT program fragment:

```forth
: BSORT ( n -- ) ... BEGIN QRD-SET QQ BEGIN INNER-LOOP Q-TEST UNTIL ... ;
```

Q-TEST implies establishing a flag for UNTIL to read. However, the written Q-TEST gives a zero flag only if QQ <> PP, otherwise no flag at all.

Secondly, constant QQ of the fragment compensates by providing a number that acts like a flag. By actual test, QQ here is always the maximum possible value (two or greater) for any number n that is two or greater.

Enclosed is a rewritten program in Forth-83 — rather than the original Forth-79 — that embodies the change of Q-TEST always supplying a flag.

Thanks for good articles like these, and please keep them coming.

Sincerely yours,
Allen Anway
Superior, Wisconsin

---

Dear Marlin:

An interesting thing happened on the way to BSORT. I keyed in the screens but, not wanting to load a random number generator (see the data.tst screen enclosed), I simply wrote INIT-DATA to fill the DATA array with memory contents of bytes zero through 100. Then I executed BSORT. But what is this? When I executed LIST-DATA, the data were only partially sorted!

In a sort of muddled attempt to find out what was going on, I executed BSORT again — and again. After a number of these, the list was properly sorted. Why did it require multiple passes?

The best I can figure it, BSORT won't sort the data into proper order on one pass if some of the data are duplicated. Listing the contents of the DATA array before sorting, I noted many duplications.

The key to this problem is in the word COMPARE-AND-SWAP, which calls SWAP-DATA if one item of data is greater than the other when the two comparison keys generated by INNER-LOOP are used to index into the DATA array. Envision a sequence of data that gets swapped because 330 is found to be larger than 225 (the second item), but another pass will be required to make the final swap, producing the order ... 330, 225, 225, 225 ....

The listings of the files batcher.blk and data.tst offer one way of fixing the problem. I decided to use variables instead of constants, as I don't think the time savings is significant in my application. I also used F83's DEFER-IS to handle the forward reference to KC.

The fix I finally decided on uses SORT-FLAG to hold a flag that forces BSORT to repeat its execution of the original loops until a pass is made that requires no data swaps. If it is known that the data to be sorted will not have duplications, there is no problem. Otherwise, the definition of SWAP-DATA, or whatever definition replaces it, must set SORT-FLAG "on" when it executes. Of course, a new outer loop for BSORT will be needed in that case. Notice that BSORT sets SORT-FLAG OFF on each pass of the new outer loop; BSORT may be left as it is and the phrase SORT-FLAG ON put into SWAP-DATA, or not, depending on whether it is needed.

I assumed a purpose (and a definition) for the word **. I have seen it defined elsewhere to produce altogether different results, but this seemed to make sense, at least to me; so, a reminder to authors: please define your non-standard words, even if they are deemed to be fairly well known.

Sincerely,
Gene Thomas
Central Arkansas FIG Chapter
Little Rock, Arkansas

Forget Automatically

When you are editing definitions in a block, the load-test-edit cycle can be speeded up by automating the FORGETting of the first word in a block. This can be done by typing CREATE TASK
**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

---

**Anway's Screens**

SCREEN # 080
0 CONSTANT TT 0 CONSTANT RR
0 CONSTANT DD 0 CONSTANT PP
0 CONSTANT NN 0 CONSTANT QQ
2DROP CONSTANT KC

: 2**N 1 SWAP SHIFT ; ( n --- 2**n )

( index1\index2 --- )

: KEY_COMP KC EXECUTE ;

: SEL_T NN 15 0 DO DUP I 2**N <= IF DROP I LEAVE THEN LOOP 1- 14 MIN ( --- ) [ ' TT 2+ ] LITERAL ! ;

( --- )

: INNER-LP NN DD - 0 DD I PP AND RR = IF I DUP DD + KEY_COMP THEN LOOP ;

: Q-TEST QQ PP = DUP 0= ( --- flag )
IF QQ PP - [ ' DD 2+ ] LITERAL !
QQ 2/ [ ' QQ 2+ ] LITERAL !
PP [ ' RR 2+ ] LITERAL !
THEN ;

---> names TT etc from Knuth

SCREEN # 081

: QRD-SET ( --- )
TT 2**N [ ' QQ 2+ ] LITERAL !
0 [ ' RR 2+ ] LITERAL !
PP [ ' DD 2+ ] LITERAL ! ;

: BSORT ( n --- )
(n ) [ ' NN 2+ ] LITERAL ! SEL_T
TT 2**N [ ' PP 2+ ] LITERAL !
BEGIN QRD-SET
BEGIN INNER-LP Q-TEST
UNTIL PP 2/ DUP
[ ' PP 2+ ] LITERAL ! 0=
UNTIL ;

---

**FORTH-83**

---> Allen Anway, Superior, WI, 1-9-87
Batcher's sort--------John Konopka
Mitaka Shi, Japan
Forth Dimensions, Nov-Dec 1986, page 39
Knuth, Art of Computer Programming,
Vol 3, pp 111-122, Addison-Wesley 1973

To use, define SWAPPER ( ind1\ind2 --- )
later, to check and perhaps swap. Then,
' SWAPPER ' KC 2+ ! ( n --- ) BSORT

---

at the start of an editing session, and writing
FORGET TEST CREATE TASK
at the top of the block. When editing of the block is complete, the references to TASK can be erased. Even the manual insertion and deletion of marker words like TASK can be avoided if LOAD and --> are redefined so that they do the job.

The new LOAD constructs a distinctive name for the block to be loaded, and looks for a word with that
As presented by John Konopka in FD Vol 8/4, BSORT will not correctly sort on one pass if there are data-duplicates in the list to be sorted. The number of passes required depends on how many duplicates there are of the same number, and with how many different numbers that occurs. Here, a sort flag is used to force BSORT to repeat until a pass with no data swap is made.

```
0 BSORT fixed.  FB3 2.0.1  
1 As presented by John Konopka in FD Vol 8/4, BSORT will not
2 correctly sort on one pass if there are data-duplicates in the
3 list to be sorted. The number of passes required depends on how
4 many duplicates there are of the same number, and with how
5 many different numbers that occurs. Here, a sort flag is used
6 to force BSORT to repeat until a pass with no data swap is made
7 (repeat while sort-flag is on)

8 BSORT SORT-FLAG ON

9 BEGIN SORT-FLAG @ WHILE DUP SORT-FLAG OFF

10 NN !: SELECT-T TT @ 21#N PP !

11 BEGIN QRD-SET QQ @

12 BEGIN INNER-LOOP Q-TEST UNTIL

13 PP @ 2/ DUP PP ! 0=

14 REPEAT DROP ;

15

Central Arkansas Forth Interest Group

\ count count executions of "xxx" to see how many passes were made

1

0 BSORT

1 DEFER KC ' NOOP IS KC VARIABLE MN VARIABLE RR

2 VARIABLE QQ VARIABLE DD VARIABLE PP VARIABLE TT

3 VARIABLE SORT-FLAG

4 KEY-COMpare KC ;

5 KEY-COMpare call the compare-and-swap definition thru KC

6 21#N (n -- n n) DUP @;

7 21#N square n

8 1-14 MIN TT !;

9 INNER-LOOP NM @ DD @ = DUP @ and RR @ =

10 IF I DUP @ * KEY-COMpare THEN LOOP;

11 Q-TEST QQ @ PP @ (=) IF QQ @ PP @ = DD ! QQ @ 2/ QQ !

12 PP @ RR @ 0 THEN;

13 QRD-SET TT @ 21#N QQ ! RR OFF PP @ DD !;

14

15 -->

0 BSORT test

1 VARIABLE XI VARIABLE X2 CREATE DATA 200 ALLOT

2 INIT-DATA (fill data array with memory contents)

3 DATA on my Kaypro I this ensures quite a few data

4 duplications, some numbers occurring >5 times

5 SWAP-DATA swap the data in DATA+nKey and DATA+nKey

6 SET SORT-FLAG ON if this is called

7

8

9

10 COMPARE-AND-SWAP compare the data pointed to by the keys output

11 by BSORT's inner-loop, swap if data+n is

12 data+n

13 'COMPARE-AND-SWAP IS KC

14 LIST-DATA list DATA array

15

FROM B:DATA.TST 1 LOAD

0 BSORT test

1 VARIABLE XI VARIABLE X2 CREATE DATA 200 ALLOT

2 INIT-DATA (fill data array with memory contents)

3 DATA on my Kaypro I this ensures quite a few data

4 duplications, some numbers occurring >5 times

5 SWAP-DATA swap the data in DATA+nKey and DATA+nKey

6 SET SORT-FLAG ON if this is called

7

8

9

10 COMPARE-AND-SWAP compare the data pointed to by the keys output

11 by BSORT's inner-loop, swap if data+n is

12 data+n

13 'COMPARE-AND-SWAP IS KC

14 LIST-DATA list DATA array

15

FROM B:DATA.TST 1 LOAD

0 BSORT test

1 VARIABLE XI VARIABLE X2 CREATE DATA 200 ALLOT

2 INIT-DATA (fill data array with memory contents)

3 DATA on my Kaypro I this ensures quite a few data

4 duplications, some numbers occurring >5 times

5 SWAP-DATA swap the data in DATA+nKey and DATA+nKey

6 SET SORT-FLAG ON if this is called

7

8

9

10 COMPARE-AND-SWAP compare the data pointed to by the keys output

11 by BSORT's inner-loop, swap if data+n is

12 data+n

13 'COMPARE-AND-SWAP IS KC

14 LIST-DATA list DATA array

15

FROM B:DATA.TST 1 LOAD

0 BSORT test

1 VARIABLE XI VARIABLE X2 CREATE DATA 200 ALLOT

2 INIT-DATA (fill data array with memory contents)

3 DATA on my Kaypro I this ensures quite a few data

4 duplications, some numbers occurring >5 times

5 SWAP-DATA swap the data in DATA+nKey and DATA+nKey

6 SET SORT-FLAG ON if this is called

7

8

9

10 COMPARE-AND-SWAP compare the data pointed to by the keys output

11 by BSORT's inner-loop, swap if data+n is

12 data+n

13 'COMPARE-AND-SWAP IS KC

14 LIST-DATA list DATA array

15

FROM B:DATA.TST 1 LOAD

0 BSORT test

1 VARIABLE XI VARIABLE X2 CREATE DATA 200 ALLOT

2 INIT-DATA (fill data array with memory contents)

3 DATA on my Kaypro I this ensures quite a few data

4 duplications, some numbers occurring >5 times

5 SWAP-DATA swap the data in DATA+nKey and DATA+nKey

6 SET SORT-FLAG ON if this is called

7

8

9

10 COMPARE-AND-SWAP compare the data pointed to by the keys output

11 by BSORT's inner-loop, swap if data+n is

12 data+n

13 'COMPARE-AND-SWAP IS KC

14 LIST-DATA list DATA array

15

FROM B:DATA.TST 1 LOAD

0 BSORT test

1 VARIABLE XI VARIABLE X2 CREATE DATA 200 ALLOT

2 INIT-DATA (fill data array with memory contents)

3 DATA on my Kaypro I this ensures quite a few data

4 duplications, some numbers occurring >5 times

5 SWAP-DATA swap the data in DATA+nKey and DATA+nKey

6 SET SORT-FLAG ON if this is called

7

8

9

10 COMPARE-AND-SWAP compare the data pointed to by the keys output

11 by BSORT's inner-loop, swap if data+n is

12 data+n

13 'COMPARE-AND-SWAP IS KC

14 LIST-DATA list DATA array

15

FROM B:DATA.TST 1 LOAD

0 BSORT test

1 VARIABLE XI VARIABLE X2 CREATE DATA 200 ALLOT

2 INIT-DATA (fill data array with memory contents)

3 DATA on my Kaypro I this ensures quite a few data

4 duplications, some numbers occurring >5 times

5 SWAP-DATA swap the data in DATA+nKey and DATA+nKey

6 SET SORT-FLAG ON if this is called

7

8

9

10 COMPARE-AND-SWAP compare the data pointed to by the keys output

11 by BSORT's inner-loop, swap if data+n is

12 data+n

13 'COMPARE-AND-SWAP IS KC

14 LIST-DATA list DATA array

15

FROM B:DATA.TST 1 LOAD

0 BSORT test

1 VARIABLE XI VARIABLE X2 CREATE DATA 200 ALLOT

2 INIT-DATA (fill data array with memory contents)

3 DATA on my Kaypro I this ensures quite a few data

4 duplications, some numbers occurring >5 times

5 SWAP-DATA swap the data in DATA+nKey and DATA+nKey

6 SET SORT-FLAG ON if this is called

7

8

9

10 COMPARE-AND-SWAP compare the data pointed to by the keys output

11 by BSORT's inner-loop, swap if data+n is

12 data+n

13 'COMPARE-AND-SWAP IS KC

14 LIST-DATA list DATA array

15

FROM B:DATA.TST 1 LOAD

0 BSORT test

1 VARIABLE XI VARIABLE X2 CREATE DATA 200 ALLOT

2 INIT-DATA (fill data array with memory contents)

3 DATA on my Kaypro I this ensures quite a few data

4 duplications, some numbers occurring >5 times

5 SWAP-DATA swap the data in DATA+nKey and DATA+nKey

6 SET SORT-FLAG ON if this is called

7

8

9

10 COMPARE-AND-SWAP compare the data pointed to by the keys output

11 by BSORT's inner-loop, swap if data+n is

12 data+n

13 'COMPARE-AND-SWAP IS KC

14 LIST-DATA list DATA array

15

FROM B:DATA.TST 1 LOAD

0 BSORT test

1 VARIABLE XI VARIABLE X2 CREATE DATA 200 ALLOT

2 INIT-DATA (fill data array with memory contents)

3 DATA on my Kaypro I this ensures quite a few data

4 duplications, some numbers occurring >5 times

5 SWAP-DATA swap the data in DATA+nKey and DATA+nKey

6 SET SORT-FLAG ON if this is called

7

8

9

10 COMPARE-AND-SWAP compare the data pointed to by the keys output

11 by BSORT's inner-loop, swap if data+n is

12 data+n

13 'COMPARE-AND-SWAP IS KC

14 LIST-DATA list DATA array

15

FROM B:DATA.TST 1 LOAD

0 BSORT test

1 VARIABLE XI VARIABLE X2 CREATE DATA 200 ALLOT

2 INIT-DATA (fill data array with memory contents)

3 DATA on my Kaypro I this ensures quite a few data

4 duplications, some numbers occurring >5 times

5 SWAP-DATA swap the data in DATA+nKey and DATA+nKey

6 SET SORT-FLAG ON if this is called

7

8

9

10 COMPARE-AND-SWAP compare the data pointed to by the keys output

11 by BSORT's inner-loop, swap if data+n is

12 data+n

13 'COMPARE-AND-SWAP IS KC

14 LIST-DATA list DATA array

15
name in the dictionary. If it is found, \texttt{FORGET} is used to delete it and its successors. Then the name is used to \texttt{CREATE} a new null word, and the block is loaded in the usual way.

The definitions given here assume that the terminal input buffer can be temporarily relocated. This would not be necessary in a version of Forth in which \texttt{FORGET} and \texttt{CREATE} can be factored into two parts: a string-grabbing part and a part that refers to the address of the string.

To recover the space occupied by all the null words created by the new \texttt{LOAD}, \texttt{FORGET} the new \texttt{LOAD} and reload everything with the old \texttt{LOAD}.

Philip Bacon
Gainesville, Florida

\textbf{Swift 6502 Multi-Tasking}

Editor,

Richard Rooney wrote to you (FD VIII/6) asking whether anyone had implemented the Laxen model multitasker on a 6502. Yes.

Six months ago I, too, worked through the Laxen tutorial. Of course, the crux of the matter is getting around the fixed return stack in page one, and the data stack in page zero (fig-FORTH). My solution, after failure with block moves (too slow), was to add a little hardware. The "little hardware" is a stack segmentation scheme. Use a PIA to switch in a new page zero/one pair for the next task.

Overhead? Two additional assembly instructions. Very fast. This is very easy to do on the old OSI Challenger III system, because of the presence of pseudo-address lines A16-A19, which are PIA controllable. I had to modify an extra memory board, which became \textit{stack memory}. This took only two ICs and about 25 jumpers. As Richard Rooney guessed, you use the \texttt{BRK} in place of Laxen's \texttt{RST}. My 6502 fig-FORTH multi-tasker handles 16 tasks. And it's fast.

I found Laxen's article to be very rewarding. So get those back issues and work through it!

Best regards,
Dale King
Leonard, Texas

\textbf{Searching for F83}

Dear FIG:

When I received the November/December issue of \textit{Forth Dimensions}, I was writing a full-screen, memory-mapped editor for Laxen and Perry's F83. I had just gotten to the point where I was able to use my new editor to add the bells and whistles — one of which was the search function. I had not intended to rewrite the one provided, but Mr. Bill Zimmerly had submitted a fast, assembly language version of \texttt{SEARCH}. This got me interested, so I studied it with the intention of replacing the high-level \texttt{SEARCH} provided in F83.

I wish to address three points: 1. Brother Zimmerly asks, in the shadow screen accompanying his code, "How could it be faster?" I can make it faster (and smaller, too). 2. Mr. Zimmerly's code does not return the same results as the \texttt{SEARCH} provided in F83 — he returns an address and a flag, while F83 returns an offset and a flag. (3) Mr. Zimmerly's \texttt{SEARCH} is not case sensitive/neutral, as is Laxen and Perry's version.

Screen 15 is Mr. Zimmerly's, and compiles to 87 bytes, including headers for \texttt{(FIND1)} and \texttt{SEARCH}. If meta-compiled so that \texttt{(FIND1)} has no header in the new system, this can be reduced to 75 bytes.

Screen 16 is my modification of Mr. Zimmerly's \texttt{SEARCH}. It compiles to 67 bytes, and I claim it will run faster. First, I eliminated the \texttt{PUSH} and \texttt{POP} of the BP register, which serve no purpose since BP is not used and, therefore, is not changed. This saves four bytes. Next, I moved the exit code from a labeled routine into the \texttt{SEARCH} code, eliminating one jump for each execution of \texttt{SEARCH}. Finally, I moved the jump from the end of the routine to \texttt{HERE} (top of loop) in the IF statement. This will eliminate a jump every time there is a match on the first character but failure to match on the whole string.

(Continued on page 36.)
Although Forth’s interactive environment simplifies testing and debugging, there are times when a word or component is difficult to verify. Test prints can be used to clarify what is happening within a word at run time; but it is tedious to insert these print statements when they are needed, and then to either delete or comment them out when they are not.

The words described below set up a test environment that allows one to choose whether or not messages will be printed at run time, without modifying source screens. I have implemented this environment in polyFORTH on an IBM-AT, MVP-FORTH on a Texas Instruments Professional, and SuperFORTH on a Commodore-64. This article describes the environment, how to implement it, and how it may be used.

Test Mode Control
The first step is to establish a means of controlling the test environment. Screen 10 defines the constant TEST? which indicates test status. It is defined as a constant rather than as a variable, since it is to be used far more often that it is to be changed. The constants YES, NO, TRUE, and FALSE allow some variation in phrasing commands. The word .L will display the value of a logical flag as either YES or NO, so the effect of TESTING? is:

TESTING? YES OK

The word TESTING will change the value of TEST?. For example, the phrase YES TESTING will set the testing mode.

Conditional Display Words
The test words defined in screen 11 will display information if the test mode is set, but will have no effect if it is not. The words T., TCR, and T.S cover situations I encounter frequently. Their definition simply makes subsequent words more compact and easier to read. However, TEST" is a bit more complicated than the others.

Certainly, one could set up conditional prints as needed, for example:

... TEST? IF ." Two Real Roots " THEN ...

The message would appear only if the test mode is set, but this takes up more room in each definition, and is far less elegant, than:

... TEST" Two Real Roots " ...

The effect of TEST" is much like that of ABORT", except that ABORT is not called. Its definition was adapted from that of ABORT" and ." as described in All About Forth (Glen B. Haydon, Mountain View Press). Basically, two words are needed: <TEST1> to conditionally print a string compiled into a word’s definition, and TEST" to compile that string.

The immediate word TEST" will compile into the definition <TEST"> and the string following TEST" in the input stream. For example, suppose the phrase below is entered:

... TEST" PHASE A ... "...

After checking the state, TEST" compiles the code field address (CFA) of <TEST"> into the definition. The ASCII code for " is 34, so the phrase 34 WORD will place the string PHASE A into the dictionary. But WORD will not affect the dictionary pointer (DP), although it does leave its current value (HERE) on the stack. The phrase C@ 1+ ALLOT then increments the dictionary pointer to the first cell beyond the end of the string. This series of events is depicted in Figure One.

<TEST"> expects a string to be stored in the word’s definition immediately after its address. Note that the string starts at the same address as that pushed on the return stack when <TEST"> is called.

<TEST"> first sets up the string for TYPE, then adjusts the return address so that it points to the first address beyond the string. If this were not done, the interpreter would return to the count byte and first text byte of the string, with disastrous results. Then, if the test mode is set, <TEST"> will display the compiled string; otherwise, the string’s address and length count are dropped from the stack. This action is depicted in Figure Two.

The version of <TEST"> shown has worked on all three Forth systems tested, but TEST" is somewhat system dependent, so it should be implemented...
with care. The version shown works in MVP-FORTH and SuperFORTH-64, but the string is compiled in the polyFORTH version of \texttt{TEST} with the phrase \texttt{STRING}, which replaces lines 11 and 12 of screen 11.

If you want to try this in some other version of Forth, be aware that \texttt{WORD} does not work the same in all dialects. Some versions leave \texttt{HERE} on the stack and others do not. In addition, watch that \texttt{TEST"} and \texttt{<TEST"}> agree on where the next address after the string is stored.

A lot of my personal taste goes into screens 10 and 11. There are many synonyms for \texttt{TRUE} and \texttt{FALSE}, and you may wish to add to or delete from the list of test words. But there are three suggestions I would make. First of all, set up \texttt{.L} to print the same number of characters in both the true and false case. This makes it easier to use \texttt{.L} to set up tables that look nice. Secondly, be sure that all \texttt{TEST} words are stack neutral. Note, for example, that \texttt{T} duplicates the top value before printing. The final suggestion is to set up \texttt{TEST?} as a constant. Not only will you save two bytes in every defining screen and two in each definition, but you won't have to remember whether to use \texttt{&} or \texttt{C@}.

\textbf{A Better Way}

Nifty as this system is, there are two major drawbacks to its use. First of all, although nothing is displayed when the test mode is not set, \texttt{TEST?} is still being checked by each test word. \texttt{TEST"} is a real time consumer, since even when \texttt{TEST?} is false, \texttt{<TEST"}> must still manipulate return addresses to avoid a system crash. The execution time for \texttt{TEST"} when \texttt{TEST?} is false is about 0.1 msec. on the IBM-AT, and 18.5 msec on the Commodore-64, which is enough time to affect some applications.

Secondly, when one is sure that a set of words no longer needs testing, its definitions are still cluttered up with the test prints, which uses dictionary space needlessly. Of course, you could go back and comment out all the test prints, but this is tedious and hard to change back, if an unexpected bug arises.

Screens 12 and 13 display more advanced versions of the words defined in screen 11. If \texttt{TEST?} is true at compile time, then these words work exactly as their earlier versions. If \texttt{TEST?} is false, however, nothing at all is compiled into the definition.

The method used for most of the words is straightforward. For example, \texttt{<TCR> in screen 12} is defined exactly as \texttt{TCR was in screen 10}. Then, the immediate word \texttt{TCR} is defined to compile \texttt{<TCR>} into a word only if \texttt{TEST?} is true. This pattern can be followed for any of the other words, except \texttt{TEST"}.

The only problem with \texttt{TEST"} is how to deal with the string following it in the input stream if \texttt{TEST?} is false. The definition of \texttt{TEST"} in screen 13 functions just the same as the earlier version in screen 11 if \texttt{TEST?} is true, but uses \texttt{WORD} to move the interpreter beyond the string if \texttt{TEST?} is false. The polyFORTH version also uses \texttt{WORD} to do this, but has no \texttt{DROP} in line 10, since its version of \texttt{WORD} does not leave \texttt{HERE} on the stack.

\textbf{Using the System}

\texttt{TEST?} may be checked at both compile time and run time. If \texttt{TEST?} is true at compile time, conditional code is inserted so that you can turn displays on and off at run time. However, if \texttt{TEST?} is false, the code is not inserted and test displays will not appear at run time, regardless of the value of \texttt{TEST?}.

Suppose an application consists of five screens, four of which are thoroughly debugged, and one (screen 23) that is not. A load screen might appear, as follows:

\texttt{NO TESTING 22 LOAD YES TESTING 23 LOAD NO TESTING 24 26 THRU}

The result would be that any test code in screens 22, 24, 25, or 26 would not be compiled, but test code in screen 23 would. Thereafter, whenever \texttt{TEST?} is true, the test displays from screen 23 would appear.
SCREEN #10
0) ( TEST ENVIRONMENT CONTROL JPM 06NOV85)
  1) -1 CONSTANT TRUE TRUE CONSTANT YES
  2) 0 CONSTANT FALSE FALSE CONSTANT NO
  3) : L ( LF) IF " YES" ELSE " NO" THEN ;
  4) YES CONSTANT TEST?
  5) : TESTING? ;TEST? .L ;
  6) : TESTING ( LF) (') TEST? ! ; EXIT
  7) .L (LF) IF "YES" ELSE "NO" THEN ;
  8) IT IS SET UP TO PRODUCE THE SAME WIDTH OF OUTPUT IN EITHER CASE
  9) IN ORDER TO SIMPLIFY SETTING UP TABULAR DISPLAYS.
 10) TEST? IS USED TO SIGNAL WHETHER OR NOT THE TEST MODE IS SET.
 11) TESTING? WILL REPORT THE CURRENT STATE, TESTING WILL CHANGE IT.
 12) E.G. NO TESTING RESETS THE TEST MODE.
 13)
 14)
 15)

SCREEN #11
0) ( TEST* ETC, V1 - RUN TIME CHECKING OF TEST? JPM 06NOV85)
 1) TCR TEST? IF CR THEN ;
  2) : T. TEST? IF DUP . THEN ;
  3) : T.S TEST? IF .S THEN ;
  4) : TEST> ( CONDITIONAL PRINT OF A COMPILED STRING )
  5) R@ COUNT ( SET UP THE STRING FOR TYPE )
  6) DUP 1+ R> + R. ( ADJUST RETURN ADDRESS)
  7) TEST? IF TYPE ELSE 2DROP THEN ;
  8) TEST* ( COMPILED STRING AND <TEST*> )
  9) : ?OMP TEST? IF COMPILE <T.> THEN ; IMMEDIATE
 10) COMPILE <TEST*> ( RUN-TIME CODE FOR TEST? )
 11) 34 WORD ( DELIMITER IS " )
 12) @ 1+ ALLOT ; ( ADVANCE HERE TO THE END OF THE STRING )
 13) IMMEDIATE EXIT
 14)
 15)

SCREEN #12
0) ( SIMPLE TEST WORDS CHECK TEST? AT COMPILE TIME JPM 06NOV85)
 1) : <TCR> TEST? IF CR THEN ;
  2) : TCR ?OMP TEST? IF COMPILE <TCR> THEN ; IMMEDIATE
  3) : T. TEST? IF DUP . THEN ;
  4) : T.S TEST? IF COMPILE <T.S> THEN ; IMMEDIATE
  5) : T.S ?OMP TEST? IF COMPILE <T.S> THEN ; IMMEDIATE EXIT
  7)
  8)
  9)
 10)
 11)
 12)
 13)
 14)
 15)

SCREEN #13
0) ( TEST", VII - CHECKS TEST? AT COMPILE TIME JPM 04MAY86)
 1) : <TEST*> ( CONDITIONAL PRINT OF A COMPILED STRING )
  2) R@ COUNT ( SET UP THE STRING FOR TYPE )
  3) DUP 1+ R> + R. ( ADJUST RETURN ADDRESS)
  4) TEST? IF TYPE ELSE 2DROP THEN ;
  5) : TEST* ( CONDITIONAL COMPILATION OF A STRING AND <TEST*> )
  6) ?OMP TEST? IF
  7) COMPILE <TEST*> ( RUN-TIME CODE FOR TEST" )
  8) 34 WORD @ 1+ ALLOT ( COMPILE THE STRING )
  9) ELSE
 10) 34 WORD DROP ( DISCARD THE STRING )
 11) THEN ; IMMEDIATE EXIT
 12)
 13)
 14)
 15)
From time to time, there have been various proposals for dealing with internal, or local, names. By these, I mean the names of words that are only used internally in an application, never called from outside. A particularly nice scheme was presented by Dewey Val Schorre, back in *Forth Dimensions II/5* ("Structured Programming by Adding Modules to Forth"). Dewey brought the notion of a "module," and presented three words (INTERNAL, EXTERNAL, and MODULE) to implement this notion. Each module is a self-contained unit, and communicates with the outside world by means of words defined between EXTERNAL and MODULE (as well as any constants or variables defined before INTERNAL). Words defined between INTERNAL and EXTERNAL may be referenced from within the module, but are not accessible from outside. Each module may be a complete application or a building block in a larger application.

Dewey's implementation of these words was simple but effective, involving the replacement of a dictionary link to remove the internal words from the search chain. Dewey acknowledges that such a simple implementation does not allow the dictionary space savings that could be possible — the headers of the internal words are unused after MODULE has been reached, but remain in the dictionary. (Note that the parameter fields of these definitions are needed for execution, but not the headers.)

Perhaps one reason why Dewey's words have not gained a wider acceptance is that, since his article, more elaborate vocabulary structures have evolved, and the effect of Dewey's words can be obtained by using vocabularies in an appropriate way. I have always felt, though, that Dewey's words state more clearly what is going on, since vocabularies have many other uses besides this particular one.

At this point, I am not going to attempt a full justification of a modular style of programming, which is now accepted as an essential discipline in the construction of any significant application. Let me just say that I have found it very useful to have this discipline enforced (the temptation to lapse into spaghetti code is always there). If I try to call an internal name from outside a module, it is good to have the system throw it back in my face. It means I probably have not understood either the problem or my "solution" to it! It also means that if I want to change the specification of an internal word, I know I don't have to look very far to find all the references to it.

Let us now turn to the question of the potential space savings. Note that we are not realizing this with modules implemented by means of vocabularies, any more than we could with Dewey's implementation of his words. That is the reason for the code presented here. I have found that space savings on the order of 20% is possible — the shorter the definitions, the greater the savings. This means that good Forth programmers will benefit more from this code than bad ones (bad programmers, please stop reading). This code will do even more — it will allow a whole module, parameter fields and all, to be loaded temporarily and then dismissed from memory, while leaving subsequent definitions intact. It is very useful to be able to do this with an assembler, as one obvious example.

It turns out that Dewey's three words are more suitable here than if we tried to do the job with vocabularies. That would not be impossible, just more difficult. Moreover, Dewey's words are very clear and say exactly what they mean — unlike, say, FORGET-VOC. Perhaps they may find a new lease on life with this implementation.

Working in this area inevitably brings us to various implementation dependencies, since the Forth standard quite rightly leaves unspecified such details as the internal structure of the dictionary. So what we will do is focus on one specific implementation model, Laxen and Perry's F83. This code should be easily adaptable to any Forth-83 system, however, and probably to many other Forts as well. On screen 67 we give the definitions of most of the non-standard F83 words we use, in case you need them. Unfortunately, there are several other words for which we can't do this without getting bogged down in a mass of irrelevant detail. But we will now try to give enough information for our purposes here.

Firstly, the words DEFER and IS, respectively, create and redirect an execution vector. These words have been fully described by Henry Laxen in two "Techniques Tutorials" *(FD III/6, V/6)* that are well worth reading. Secondly, a
couple of non-standard words are called from FIND, which we redefine on screen 72. However, the change we make to the definition of this word is very minor, and comes right at the end. Thus, we can ignore the details of the workings of FIND. In fact, even if you don’t have F83, the change to FIND will probably be exactly the same. Thirdly, we need to refer here to the F83 word HEADER. The change is simply to rename it (HEADER), so again we don’t need to go into the details of its workings. We have more to say about it below.

Our scheme operates by separating the headers of the internal words. These go into a special area, the separated heads area (SH, for short) and are later forgotten. This is not as simple as it sounds, since internal words will have external words compiled after them, at which time the internal names must still exist. Later, if we forget the internal names by simply chopping the dictionary, we will lose the external names, too. Rather, we must follow the dictionary links and unlink any internal names, leaving the rest of the dictionary intact. We provide here a new forgetting word to do this — the regular FORGET need not be changed.

Now we will go into a bit more detail. The SH area has its own "dictionary pointer," the variable SH-DP. The base and top of the area are pointed to by two other variables, SH-DPO and SH-TOP. I am defining these as system variables here, although you may want to make them user variables if you do any multitasking on your system.

Each header in the SH area looks exactly like a normal header. The difference is that, instead of being followed by a code field, it is followed by a pointer to the code field. The code field itself sits in its usual position in the dictionary (and, of course, has no header in front of it). Here is how we set up this kind of header. In F83, headers are laid down by the word HEADER, which returns with the dictionary pointer DP pointing to where the code field will go. CREATE calls HEADER, then stores the code field. Our modification to HEADER is simply to make it vectored.

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, 80186, 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, Trisse Neustadt, 7051-1865
UK: System Science Ltd., London, 01-248-0962
France: Micro-Sigma S.A.R.L., Paris, (1) 42.05.61.61
Japan: Southern Pacific Ltd., Yokohama, 045-314-9514
Australia: Wave-onic Associates, Wilson, W.A., (09) 451-2046

Volume IX, Number 2  13  Forth Dimensions
So rename HEADER to (HEADER) and follow it with the code:

```
DEFTER HEADER' (HEADER) IS HEADER
```

When internal words are being compiler, HEADER is redirected to execute SEP-HEADER, which lays down a separated header and pointer to the code field. As required, we leave DP pointing to where the code field will go. SEP-HEADER is defined on screen 67. Notice that this word calls (HEADER) to do most of the work.

The F83 word NAME> takes the address of a name field and returns the corresponding compilation address (in F83 and most Forths, this is the same as the address of the code field). This word must be redefined so that, if the name is in the SH area, the pointer following the name is used to obtain the compilation address. This we do on screen 66. Other Forths should use either this word or something very similar. Any other words that go from a field within a header to the corresponding code field or parameter field will also need to be checked. If they use NAME> (or its equivalent) to make the transition, there's no problem. In F83, the only word in this category that doesn't use NAME> is FIND. As we mentioned above, the necessary change to this word is very minor. For completeness, we give the whole definition on screen 72.

This brings us to the new forgetting word. One of its features is that it takes a pair of addresses and forgets everything between them. So we will call it !FORGET! (pronounced "forget between"). Its definition is given on screen 68. Note the use it makes of the two other words on that screen, UNLINK and TRIM. TRIM, in its turn, uses UNLINK — it is UNLINK that does all the hard work.

F83 has a multi-thread dictionary structure, with the constant #THREADS defining the number of threads. TRIM contains a DO loop over the threads, calling UNLINK for each one. If your system does not have a multi-thread dictionary, removing the DO loop from TRIM may be sufficient.

!FORGET! can take a few seconds to execute, depending on your processor. This may not matter, as it won't be used with high frequency. However, if you can rewrite UNLINK in code for your particular processor, so much the better. This may save a few bytes of memory as well.

If you want to, you can replace the regular F83 forgetting scheme with this new one. (Thus, our TRIM has the same name as its F83 counterpart.) Simply redefine

```
: (FORGET) DUP -1
!FORGET! DP ! ;
```

and replace TRIM with our new one. FORGET itself need not be altered. The advantage of doing this replacement is to save space, since our forgetting scheme is really an extension of the old one and covers much of the same ground. It works identically if there are no SH names, and the high limit is set to the highest address. If UNLINK is written in code, FORGET should run as fast as before.

The word SET_SH_AREA (screen 67) sets up the SH area by initializing the three pointers to this area. As written, it sets up 2000 bytes for the area in high memory, and 200 bytes below the current top of the parameter stack (which grows downwards). If this is not suitable for your system, change it as necessary. The number 2000, set up as the constant SH_AREA_SIZE, I found to be sufficient for my needs. If you have a big application, you may need to increase this number. The variable SH-MAX keeps track of the maximum size required so far for the SH area, so you can find by experimentation what size you need. Separate modules (i.e., not nested) reuse the same space in the SH area, to minimize the required size.

Modules may be nested in the fashion:

```
INTERNAL...INTERNAL...EXTERNAL...MODULE
EXTERNA...MODULE
```

This is the same as provided in Dewey's original implementation of these words. Notice that once we are EXTERNAL, we can't become INTERNAL again in the same module. This could, however, be implemented. Being such perceptive readers, you will no doubt have noticed that !FORGET! is more general than we really need. Internal names to be forgotten will always occur in a single, contiguous cluster, whereas our implementation of !FORGET! will allow them to occur randomly, anywhere in the dictionary search order. This was, in fact, easier and shorter to implement (although slower). But it does mean that we can implement the more complicated module format, if necessary, without much trouble. I haven't done it here, feeling there may well be different schools of thought as to whether this feature will be desirable. Code is probably easier to follow if all EXTERNAL definitions are together, and this can usually be arranged quite easily. But go ahead and implement the more complicated format if you want to.

Our scheme for the temporary loading of entire modules (screen 71) is a straightforward application of !FORGET!. The word TEMP_MODULE saves DP on the stack, then resets it to halfway between where it was and the bottom of the SH area. You then load the temporary module, and use END_TEMP to set DP back to where it was. Hopefully, there is enough room between there and the temporary module for the definitions you now want to load. When this is done, FORGET_TEMP forgets the temporary module. If the temporary module has defined any vocabularies, remember to remove them from the search order before FORGET_TEMP — unless you find crashes entertaining! A very simple example is shown on screen 75.

Michael Hore is a missionary Bible translator with a programming background. He works among a remote group of Aborigines and uses an LSI 11/2. He is progressively moving all his software over to Forth, "...the way computers were meant to be programmed."
0 \ Separate header area
1 VARIABLE SH-DPO
2 VARIABLE SH-DP
3 VARIABLE SH-TOP
4 VARIABLE SH-EXTNL?
5 2000 CONSTANT SHAREASIZE
6 VARIABLE EXTNL?
7 VARIABLE SH-MAX

SEPARATE HEADER AREA

5 \ SH area, cont.
1 : WITHIN? ( n lo hi -- n f ) OVER --R OVER SWAP - R SWAP U( NOT
2 ।
3 ।
4 ।
5 \SEP_HDR? ( addr -- addr f ) SH-DPO \ SH-TOP \ WITHIN? ;
6 \ ?SEP_HDR? ( ?acf -- acf ) SEP_HDR? IF \ THEN ;
7 \ NAME) ( anf -- acf )
8 \ 1 TRAVERSE I+ ?SEP_HDR? ;
9 ।
10 \ Name)
11 \ ( name -- ) HERE ( save ) SH-HERE DP !
12 \ ( HEADER) ( saved dp ) DUP , DP @ SH-DP ! DP ! ;
13 \ SET_SHAREA SP 200 - DUP SHAREASIZE -
14 \ DUP HERE 200 + U( ABORT* Not enough room in memory*
15 \ DUP SH-DPO ! SH-DP ! SH-TOP ! ;
16 \ ?SET_SHAREA SH-DPO @ 0= IF SET_SHAREA THEN ;
17 \ 0 CONSTANT False
18 \ -1 CONSTANT True
19 \ ON ( addr -- ) TRUE SWAP ! ;
20 \ OFF ( addr -- ) FALSE SWAP ! ;
21 \ ?PAIRS ( n -- ) = NOT ABORT* Unbalanced structure* ;

SH-DPO points to the base of the SH area.
SH-DP Current SH area pointer.
SH-TOP Points to the top of the SH area.

Note: these three are all zero if the SH area has not been
initialized.

SHAREASIZE Size we allocate for SH area. Alter if necessary.

EXTNL? A variable to allow us to check that the order
EXTERNAL...MODULE is always followed (otherwise we'd crash).

SH-MAX Records the maximum SH space used so far.

SH-HERE Fetches the current SH area pointer.

SEPARATE HEADER AREA

May85 MRH

Volume IX, Number 2

Forth Dimensions
UNLINK ( lo hi 1st-link -- lo hi 1st-link ) DUP )R
BEGIN ?DO
WHILE ( DUP )R ( BEGIN 2 PICK 2 PICK WITHIN? WHILE 2 ) REPEAT
DUP )R !
REPEAT )R ;

TRIM ( lo hi voc-link-addr -- lo hi voc-link-addr )
I @THREADS 2+ 1 LITERAL -
#THREADS 0 DO UNLINK 2+ LOOP ;

( FORGET ) ( lo hi -- ) OVER FENCE 2+ UC ABORT' Below FENCE'
VOC-LOCAL UNLINK
BEGIN 2+CPICK WHILE TRIM REPEAT
2DROP ;

UNLINK Goes down a linked list starting with 1st-link, unlinking all links located at addresses within the range lo to hi (inclusive). This word could profitably be put into code.
TRIM For the given vocabulary, removes all words whose link fields are located within the range lo to hi. @THREADS is a constant giving the number of dictionary threads.
( FORGET ) Forgets all words whose link fields are located within the range lo to hi. DP is not changed. We first check that lo is not below where FENCE points, to guard against wiping out the system. Then we use UNLINK to remove any vocabularies that are to be forgotten. (VOC-LOCAL is the head of the vocabulary list.) Then we go down the pruned list of vocabularies and call TRIM for each one.

SAVE HEADER ( -- acct 20 ) (' HEADER )BODY 2 20 ;
INTERNAL EXTNL?
ABORT' INTERNAL follows EXTERNAL - probably MODULE omitted (*?
?SET...SHAREA SH-DP @ ( save ) SAVE HEADER
200 SH-DP 0!
'REP HEADER IS HEADER ;

INTERNAL Subsequent definitions will have separate headers.
Note we reserve 200 bytes in the SH area for EXTERNAL names, in case this is a nested INTERNAL.
Now INTERNAL is defined, we can make use of it right away.

RESTORE HEADER Restores the previous setting of HEADER.
(SH FORGET) Forgets all (SH) names above the limit, faddr.

EXTERNAL As for EXTERNAL below, but won't be accessible later, as an INTERNAL is now current. Needs to be used to make us external again, so EXTERNAL itself will be accessible.

EXTERNAL Subsequent defined names go where they were going before the last INTERNAL. These names will still be accessible after MODULE.

MODULE Forgets all names defined between INTERNAL and EXTERNAL. Names after EXTERNAL are still accessible.
0 \ TEMP_MODULE, etc.
1 0 \ TEMP_MODULE
2 \ ?SET_SHL
3 AREA HERE ( save ) SAVE_HEADER
4 \ (HEADER) IS HEADER
5 HERE SH-DPO \ OVER - 2 \ [ HEX ] 7FFE AND [ DECIMAL ]
6 + DP ! ;
7 8 \ END TEMP
8 RESTORE_HEADER DP ! ;
9 10 \ FORGET TEMP
11 HERE SH-DPO \ (FORGET) ;
12 14 MODULE
15
16 0 \ Modified FIND
17 0 \ FIND ( addr -- acf flag : addr false )
18 PRIOR OFF FALSE NVOCs 0
19 DO DROP CONTEXT I 2x + 2 DUP
20 IF DUP PRIOR \ OVER PRIOR ! =
21 IF DROP FALSE
22 ELSE OVER SWAP NVOCs 2
23 (FIND) DUP ?LEAVE
24 THEN THEN LOOP
25 DUP IF SWAP \ ?SEP_HDR) SWAP THEN ;
26 13 END
27 14 EXIT
28 15

0 \ Module example
1 0 \ INTERNAL
2 3
4 \ (CHANGE_CASE) ( c -- c' )
5 ASCII A ASCII z WITHIN? SWAP ASCII a ASCII z WITHIN?
6 ROT OR IF BL XOR THEN ;
7 8 \ EXTERNAL
9 10 \ CHANGE_CASE ( addr len -- ) BOUNDS
11 DO I C! (CHANGE_CASE) I C! LOOP ;
12 13 \ MODULE
14 15

0 \ TEMP_MODULE, etc.
1 1071 \ TEMP_MODULE MARKS the start of a module (such as the assembler) which is to be forgotten once it has finished. This word leaves DP pointing to where the module will be loaded. Currently this is half-way between HERE and the bottom of the SH area. You can change this if necessary.
2 3 1072 \ END TEMP
4 Used after the temporary module is loaded. Restores DP to its usual position.
5 FORGET TEMP
6 Used when the temporary module is no longer needed. Forgets it using (FORGET), so nothing else is affected.
7
8 0 \ Modified FIND
9 0 \ FIND ( addr -- acf flag : addr false )
10 PRIOR OFF FALSE NVOCs 0
11 DO DROP CONTEXT I 2x + 2 DUP
12 IF DUP PRIOR \ OVER PRIOR ! =
13 IF DROP FALSE
14 ELSE OVER SWAP NVOCs 2
15 (FIND) DUP ?LEAVE
16 THEN THEN LOOP
17 DUP IF SWAP \ ?SEP_HDR) SWAP THEN ;
18 13 END
19 14 EXIT
20 15

0 \ Module example
1 0 \ INTERNAL
2 3
4 \ (CHANGE_CASE) ( c -- c' )
5 ASCII A ASCII z WITHIN? SWAP ASCII a ASCII z WITHIN?
6 ROT OR IF BL XOR THEN ;
7 8 \ EXTERNAL
9 10 \ CHANGE_CASE ( addr len -- ) BOUNDS
11 DO I C! (CHANGE_CASE) I C! LOOP ;
12 13 \ MODULE
14 15

Although CHANGE_CASE uses (CHANGE_CASE), the latter is now not accessible, and its name is not taking up any memory space.

Volume IX, Number 2
17
Forth Dimensions
If you have ever played with Tinker Toys or Erector Sets, you’ll have some appreciation for the fischertechnik products. In Europe, this West German company is extremely popular for its construction kits, ranging from simple block models similar to those by Lego, through sophisticated electronic, pneumatic, and hydraulic experimentation kits. The package we are concerned with is called the “fischertechnik computing” line, consisting, at the time of this writing, of a robotic construction kit.

With this kit, you can design and build simple, two-axis robots, connect them to your computer, then program and run them. The original kits from fischertechnik use BASIC as the programming language. When we (at Parsec Research) saw the kit, we were enthralled by the possibilities it suggested. Being Forth-oriented, we naturally couldn’t help but question the choice of BASIC as a language for conducting experiments with robots. So we decided to create an “improvement” on the original. The result was PaRCL (pronounced “parkul”), Parsec Robot Control Language.

The original kits came with very little help by way of manuals. For the dedicated computer hobbyist, the program listings and minimal explanations would be adequate, but to the average consumer, the thing would be totally cryptic. Given this, I decided to write a text to help a novice computerist learn how to program the robots described in the kit. The primary consideration in designing PaRCL was this: I must be able to explain what is happening as simply as possible. If the explanation for some language feature is too cumbersome, then it probably is inappropriate to include it in the language.

Why Forth is Used

Most Forth programmers already know why their favorite language is superior to all others. Most Forth programmers are also aware of the fierce opposition that sometimes exists among others. Epithets aside, I selected Forth as the basis of a consumer-oriented product for three reasons: functionality, interactivity, and familiarity.

Forth is a functional language; that is, when you program in Forth, you create functions to express an operation you wish to perform. Thus, you can actually create a collection of specialized vocabularies to talk about the different types of things you want your program to do. This makes it very easy for me to explain to someone how to make the robot do some sequence of actions.

For example, suppose I give this description for a robotic arm:

```
EXTEND-ARM-TO-TARGET
PICK-UP-OBJECT
RETRACT-ARM
ROTATE-TO-POSITION-3
EXTEND-ARM-TO-TARGET
DROP-OFF-OBJECT
```

This pretty much says it all. Naturally, we must assume here that I’ve already described what each of the steps entails, and that the reader is acquainted enough with Forth syntax to accept the hyphenated words as single commands.

Forth’s interactivity is an immediate benefit. If I have taken the user through the steps necessary to define the words given in the above example, I can have him actually try them out immediately.

Finally, I am familiar with Forth. Given the choice, I generally prefer it to another language. Of course, given an appropriate incentive, I might consider learning COBOL...

Consumerization

When micros first became available, the language of choice (the only language, for that matter) was BASIC. Now, with Borland’s efforts, it seems that everyone is learning Pascal. So how did I expect to foist yet another language (and a peculiar one, at that) on an unsuspecting public? Well, the first problem when getting someone to try something new, is to make it palatable (or at least marginally tolerable). Thus, it was necessary to “consumerize” Forth for this application.

As I stated earlier, when designing PaRCL, my primary criterion for inclusion of any language feature was explainability. If I could explain it, it was probably usable. If I could not explain it, then it was probably useless. This meant that a lot of standard Forth words were scrapped from the PaRCL vocabulary. After all, it is intended to be used to write robot programs, not computer programs.

Most Forth books start by showing the novice programmer how to display messages and numbers. Then they jump into a discussion of the parameter stack. Now, this is the second most frequently criticized fact of Forth life: the existence of a naked stack. (The first most criticized aspect is reverse polish.) To a Forth

---

**CONSUMERIZED FORTH**

KEN TAKARA - SAN JOSE, CALIFORNIA
CALL FOR PAPERS
for the ninth annual

FORML CONFERENCE

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

Following Thanksgiving, November 27-29, 1987

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

Theme: Forth and the 32-bit Computer

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

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

Papers on other Forth topics are also welcome. Mail your abstract(s) of 100 words or less by September 1, 1987 to:

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

Completed papers are due November 1, 1987. For registration information call the Forth Interest Group business office at (408) 277-0668 or write to FORML Conference.

Asilomar is a wonderful place for a conference. It combines comfortable meeting and living accommodations with secluded forests on a Pacific Ocean beach. Registration includes deluxe rooms, all meals, and nightly wine and cheese parties.
programmer it is simply accepted, and often abused. To a novice, however, it can be a major roadblock. Since I was catering to novices, I decided to reduce the number of stumbling blocks by eliminating any mention of the stack. This meant that all our beloved stack words (DUP, SWAP, ROT, DROP, etc.) had to go.

Instead of the stack, I emphasized the use of variables. I realize the stack is intended to improve certain aspects of Forth programming. But I also know from experience that most novice programmers feel much more comfortable with variables, which they simply accept, and often abuse. In the case of a systems programmer concerned with efficiency or speed, this is a problem. But for everyone else, who cares it it's slow, as long as it works, and the programmer's intention is clear?

Naturally, for this somewhat lobotomized version of Forth, most of the disk operators, dictionary operators, double-word arithmetic, and data-conversion words were eliminated as irrelevant for simple robot control. Of course, the user should have some way to write programs and to save them, so I included a source editor and a few disk operators such as LOAD, THRU, and LIST. I also included a hard-copy word, PRINT-SCREENS.

I did use Forth screens. It was easier to explain screens than to create a file system, and people seem to get a kick out of having unrestricted access to the disk. The business of opening files, appending to files, and closing files is complicated enough, and more so to a computer novice. Also, if I supported a file system, I would have to explain how to use it, and that would have been a chore. Being lazy, I took the easy way out.

Preserved Features

So, considering what was removed, you probably wonder that I left anything Forth-like intact. I did. In fact, ParCRL uses reverse Polish without any modification. Thinking in terms of passed parameters, it is relatively easy to explain how the output from one command (I called them commands instead of words) is held by the computer, and is used as input to another command. To a complete novice, the computer is quite mysterious anyway, so I took the liberty of simply using reverse Polish without any technical explanation.

This may seem contradictory, compared to my previous concern for explainability. How can I worry about explaining everything, then decline to explain the mysteries of reverse Polish? In fact, my intention was to explain how to use ParCRL to program the robots. Since the arithmetic for this kit is relatively simple, there was no reason to bother with algebraic parsing. Most people are willing to accept a few peculiarities, as long as they are kept to a minimum.

I also kept single-word integer arithmetic. Again, for this kit, this was entirely satisfactory. The physical limits of the components are well within single-precision boundaries.

Can Forth Succeed?

It has long been the dream of Forth programmers to bring their language to the attention of the masses. We would all like to see Forth accepted as a "respectable" language for programming. Will our attempt with ParCRL successfully engender acceptance of Forth among non-Forth users? What is required for success in this effort?

I don't know whether or not Forth will ever be popularly received. I certainly didn't choose it in order to persuade anyone that it is the best language; it was simply the easiest language for presenting robotic programming.

The acceptance of Forth outside the diehard Forth community depends on two factors: the manner in which it is presented, and the receptiveness of the audience. If you try to proselytize, you will probably face a great deal of resistance. If you present it to a novice programmer with promises of effortless programming, you will surely disappoint him. And if you make exaggerated claims about Forth, you will always provoke an argument.

An experienced Fortran programmer usually has no desire to learn another language; he is already at ease with what he has, and can do whatever he needs. The same may be said of a BASIC programmer, an Ada programmer, and so on. Half of these people see no point in struggling through the entire learning process with an unfamiliar language. Another quarter of them are actively hostile to anyone who might challenge the capabilities of their favorite language. And the last quarter, interested in programming languages in general, will pick up anything promising that comes their way.

Any programmer willing to learn and use Forth should be able to appreciate its merits (and its drawbacks) without further argument. For any other programmer, it is pointless to debate the issue, just as it is pointless to discuss the merits of PL/1 with a Forth devotee.

A casual computer user will probably never gain any insight or benefit from learning Forth (or any other language, for that matter). A person who uses computers frequently might, as long as it doesn't tax his patience every time he tries to write some simple program with it. For most computer users, to have to deal with the complexities of stacks, dictionary structures, linked lists, reverse Polish notation, virtual memory, and memory-mapped I/O in order to patch together a checkbook-balancing program (yes, they still do it) is an enormous bother.

As long as Forth is merely a dedicated hacker's language, it will remain merely a dedicated hacker's language. If it is to become a user's language, as popular as BASIC, it will have to put on some respectable clothes and make itself presentable to the public. And how might we do that? Well, you'll have to figure that out for yourself.
MEMBERSHIP IN THE FORTH INTEREST GROUP

109 - MEMBERSHIP in the FORTH INTEREST GROUP and Volume 9 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 world-wide, 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 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 $42.00 per year. 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.

HOW TO USE THIS FORM

1. Each item you wish to order lists three different price categories:
   - Column 1 - USA, Canada, Mexico
   - Column 2 - International Surface Mail
   - Column 3 - International 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)

<table>
<thead>
<tr>
<th>Column 1</th>
<th>Column 2</th>
<th>Column 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>101 - Vol. 1 FORTH Dimensions (1979/80)</td>
<td>$15/16/18</td>
<td>$30/33/40</td>
</tr>
<tr>
<td>102 - Vol. 2 FORTH Dimensions (1980/81)</td>
<td>$15/16/18</td>
<td>$30/33/40</td>
</tr>
<tr>
<td>103 - Vol. 3 FORTH Dimensions (1981/82)</td>
<td>$15/16/18</td>
<td>$30/33/40</td>
</tr>
<tr>
<td>104 - Vol. 4 FORTH Dimensions (1982/83)</td>
<td>$15/16/18</td>
<td>$30/33/40</td>
</tr>
<tr>
<td>105 - Vol. 5 FORTH Dimensions (1983/84)</td>
<td>$15/16/18</td>
<td>$30/33/40</td>
</tr>
<tr>
<td>106 - Vol. 6 FORTH Dimensions (1984/85)</td>
<td>$15/16/18</td>
<td>$30/33/40</td>
</tr>
<tr>
<td>107 - Vol. 7 FORTH Dimensions (1985/86)</td>
<td>$20/21/24</td>
<td>$35/38/45</td>
</tr>
<tr>
<td>108 - Vol. 8 FORTH Dimensions (1986/87)</td>
<td>$20/21/24</td>
<td>$35/38/45</td>
</tr>
</tbody>
</table>

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, metacomilation, 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

316 — FORML PROCEEDINGS 1986 $30/33/40
Forth internals, Methods, Standards, Forth processors, Artificial Intelligence, Applications.

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 PERSONAL EXPERT SYSTEMS $19/20/29
Carl Townsend and Dennis Feucht
217 — F83 SOURCE  $20/21/30  
Henry Lazen & Michael Perry
A complete listing of F83 including source and shadow screens. Includes introduction on getting started.

218 — FOOTSTEPS IN AN EMPTY VALLEY  $25/26/35  
(NEC4000 Single Chip Forth Engine)  
Dr. C. H. Ting
A thorough examination and explanation of the NEC4000 Forth chip including the complete source to cmForth from Charles Moore.

219 — FORTH: A TEXT AND REFERENCE  $22/23/33  
Mahlon G. Kelly & Nicholas Spies
A textbook approach to Forth with comprehensive references 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

226 — 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 — LIBRARY OF FORTH ROUTINES AND UTILITIES  $23/25/35  
James D. Terry
Comprehensive collection of professional quality computer code for Forth; offers routines that can be put to use in almost any Forth application, including expert systems and natural language interfaces.

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, 2nd Edition (soft cover)  $20/21/30  
Leo Brodie
In this new edition of Starting Forth, the most popular and complete introduction to Forth, syntax has been expanded to include the new Forth-83 Standard.

246 — STARTING FORTH (hard cover)  $20/21/30  
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.

267 — TOOLBOOK OF FORTH  $23/25/35  
(Dr. Dobb's)
Edited by Marlin Ouverson
Expanded and revised versions of the best Forth articles collected in the pages of Dr. Dobb's Journal.

270 — UNDERSTANDING FORTH  $3.50/5/6  
Joseph Raymann
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  $25/28/35
(Standards Conference)
79-Standard, implementing Forth, data structures, vocabularies, applications and working group reports.

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

323 — ROCHESTER 1983  $25/28/35
(Forth Applications)
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  $25/28/35
(Forth Applications)
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  $20/21/30
(Software Management & Engineering)
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.

401 — JOURNAL OF FORTH RESEARCH V.1  $30/33/38
Robotics/Data Structures

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

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

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

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

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

409 — JOURNAL OF FORTH RESEARCH V.3 #3  $15/16/18
Application Languages

410 — JOURNAL OF FORTH RESEARCH V.3 #4  $15/16/18
Applications, Arithmetic extensions

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
**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 $20/21/22
FORTH, Inc.

**REFERENCE**

305 — FORTH 83-STANDARD $15/16/18
The authoritative 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.

**REPRINTS**

420 — BYTE REPRINTS $5/6/7
Eleven Forth articles and letters to the editor that have appeared in Byte magazine.

**ASSEMBLY LANGUAGE SOURCE CODE LISTINGS**

Assembly Language Source Listings of fig-FORTH for specific CPUs and 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 $12/13/14
"May the Forth Be With You"
Small, Medium, Large and Extra-Large

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 below are the first releases of new professionally developed Forth applications. 5 1/4" disks are IBM MSDOS 2.0 and up compatible and are compatible with Forth-83 systems listed below.

Laxen/Perry F83
LMI PC/FORTH 3.0
MasterFORTH 1.0
TaskFORTH 1.0
PolyFORTH (R) II
Macintosh 3 1/2" disks are available for MasterFORTH systems only.

Please specify disk size when ordering.

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. Those issues contain the documentation for this disk.

703 — AUTOMATIC STRUCTURE CHARTS V.3
by Kim R. Harris $40/43/45
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.

704 — A SIMPLE INFERENCE ENGINE V.4 $40/43/45
by Martin J. Tracy
Based on the Inference Engine in Winston & Horn's book of Lisp, this volume takes you from pattern variables to a complete unification algorithm. Accompanied throughout with a running commentary on Forth philosophy and style.

706 — THE MATH BOX V.6 $40/43/45
by Nathaniel Grossman
A collection of mathematical routines by the foremost author on math in Forth. Extended double precision arithmetic, a complete 32-bit, fixed-point math package and auto-ranging text graphics are included. There are utilities for rapid polynomial evaluation, continued fractions and Monte Carlo factorization.

**NC4000 SERIES**

801 — MORE ON NC4000, VOLUME 1 $10/11/14
FIG-Tree style forum on NC4000. Topics including bugs, products, tips, benchmarks, and NC4000 instruction bit patterns. Chuck Moore's teleconference. 2nd edition.

802 — MORE ON NC4000, VOLUME 2 $15/16/18
NC4000 User's Group's Newsletters. Many contributions from Chuck Moore, Rick VanNorman, C.H. Ting and many other users. Hardware enhancements, software and many; utility programs.

803 — MORE ON NC4000, VOLUME 3 $15/16/18
NC6000/5000 data sheets, quans, new DROP, DEPTH, Eaker's CASE, PICK, ROLL, floating point math packages, new power sources and A/D converters for NC4000. Many other tips.

804 — MORE ON NC4000, VOLUME 4 $15/16/18
Chuck Moore's Application Notes 1-7, Tiny Modula-2 by Lohr, F83 extensions and other tips from Bill Muench, VanNorman's screen editor, Ting's 32-bit engine design and Fourier transform.
### OFFICE USE ONLY

<table>
<thead>
<tr>
<th>By</th>
<th>Date</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Shipped by</th>
<th>Date</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>UPS Wt.</th>
<th>Amt.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TNT Wt.</th>
<th>Amt.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>USPS Wt.</th>
<th>Amt.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>BO Date</th>
<th>By</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Wt.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

### PAYMENT MUST ACCOMPANY ALL ORDERS

**CHECK ENCLOSED (Payable to: Forth Interest Group)**

**VISA**

**M/C**

**Card #**

**Expiration Date**

**Signature**

($15.00 minimum on all VISA/MC orders.)

---

**ITEM # | TITLE | AUTHOR | QTY | UNIT PRICE | TOTAL**

| 109 | MEMBERSHIP |      |     |           | SEE BELOW |

---

**SUB-TOTAL**

**ORDERS OF $50.00 OR MORE
RECEIVE A 10% DISCOUNT**

**SUB-TOTAL**

**CA. RESIDENTS ADD SALES TAX**

**HANDLING FEE**

$2.00

**MEMBERSHIP**

*NEW* ☐ *RENEWAL* $30/42

ENCLOSED IS $30/42 FOR 1
FULL YEARS DUES. THIS INCLUDES
$24/36 FOR FORTH DIMENSIONS.

---

**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**

Delivers to Alameda, Contra Costa, San Mateo, Los Angeles, Santa Cruz and San Francisco Counties, add 8½%. San Clara County, add 7%. other California counties, add 6%.
One reason Forth runs so fast is that it does not have the built-in checks of other high-level languages. But during development, this means frequent crashes, and the annoying need for hardware reset and reentry of new words into the dictionary.

This can be avoided with a temporary patch to NEXT, the Forth thread interpreter. This patch decreases execution speeds, but provides considerable execution security, removing annoying crashes.

The method is to examine CFAs for valid contents: CFA+2 for code words, and one of six values for high-level words (060D = DOCOL, 0669 = DOVAR, 067B = DOUSE, 064F = DOCON, 0AC8 = DODOE, and 020D = IDO). A jump is made to 0103H for any other.

Note: execution time is 25 to 50% longer. Remove patch after debugging.

[Editor's note: It would be interesting to make this table-driven, and to have the table automatically updated by defining words. Slowness shouldn't be too much of a drawback here.]

Patch 8080 fig-FORTH as follows: Put JMP XSPTCH at 014B.

**XSPTCH:**

MOV E,M
INX H
MOV D,M
INX H
MOV A,D
CMP D
JNZ XSPTC1
MOV A,E
CMP L
JNZ XSPTC1
DCX H
JMP 014EH

**XSPTC1:**

DCX H
MVI A,06H
CMP D
JNZ XSPTC3
MVI A,0DH
CMP E
JZ 014EH
MVI A,69H
CMP E
JZ 014EH
MVI A,7BH
CMP E
JZ 014EH

**XSPTC2:**

JMP 0103H

**XSPTC3:**

MVI A,0AH
CMP D
JNZ XSPTC4
MVI A,08H
CMP E
JNZ 0103H
JMP 014EH

**XSPTC4:**

MVI A,02H
CMP D
JNZ 0103H
MVI A,0DH
CMP E
JNZ 014EH
JMP 0103H
JMP 014EH
Thanks to Larry Forsley and company for putting on another fine Forth conference at the University of Rochester. As usual, the meeting had a number of excellent speakers and presentations, with a high level of good technical discussion. I was happy to meet some of the folks we've been typing at for a while: Bob Brown, John Hall, Dennis Ruffer, Mike Ham, Jim Callahan, Bob Davis, and others. Amazing how no one looks the way you imagine they might.

I had a brief chat with Glen Haydon about Loglan (currently being discussed on the ECFB); one interesting note was that Chuck Moore reviewed some of the early Loglan notes, and that there may have been some cross-fertilization between the two languages.

John Hall reported that the FIG GENie network was still on hold. I passed on the ECFB files (about 30 or so disks) to Dennis Ruffer for use on the FIG GENie network. It'll be nice to see this get a wider audience.

There were several talks on the massively parallel processors (11,520 16,384 separate processors) at Goddard and Lockheed, both of which are being, at least partly, programmed in Forth. George Nicol of Silicon Composers described their ability to connect multiple, Novix-based coprocessor cards to an IBM PC/AT. So far, they've been able to drive up to eight boards at a time, giving an aggregate capability of 56 MIPS (or MOPS)! In terms of raw compute power, this is easily in the mainframe range. George said that his goal is to get twenty of these things going at once so they'd be in the Crayballpark.

A few new pubs were available — a third edition of Thea Martin's excellent Bibliography of Forth References, the 1986 FORML Conference Proceedings, and an interesting, but overpriced book ($16.95 for 90 pages) called Forth: The NEXT Step by Ron Geere. I was tempted to dismiss this book because it's based on fig-FORTH and Forth-79, and a number of its examples use 6502 code definitions.

We are seeing Forth-based applications in several advanced-technology areas.

But it actually has a fair amount to offer: numerous tools on math, time/calendar manipulation, trig, large numbers, sorting, rational numbers, etc. There's some useful stuff here, but $16.95?

We had an interesting panel discussion on Forth standards. The panel consisted of Mahlon Kelly, Larry Forsley, Martin Tracy, and myself, moderated by Jim Basile. There was no general theme; people seemed generally in favor of an ANSI standard, but didn't want anyone tinkering with the language. Glen Haydon said that standards were for people who had nothing better to do. He wanted to see standardization based on common usage and indicated that he was compiling a book on this. Dick Miller simply didn't want any changes, calling himself a conservationist. Other people objected to a self-appointed group deciding on new standards. Bob Brown suggested that Forth look towards Lisp for a model of standardization, due to its many similarities. I apologize if I missed or misrepresented any of the salient points.

Portions of the following are taken directly from the session notes distributed at the conference. Many fascinating presentations are not reviewed here at all, due to space limitations, but you can find the complete proceedings in The Journal of Forth Application and Research (Volume 5, Number 1).

Sessions


“High-Density Parallel Processing: I. The Processor Array and Macro Controller; II. Software and Programming,” Nguyen, Raghaven, C. H. Ting, and Truong (Lockheed Palo Alto Research Laboratories). Another huge array of processors — this one has only (1) 11,520 individual processors. Dr. Ting noted that this one obeys the rule “Never trust a computer you can't lift,” but he seemed to conclude that meant without chassis and power supply.

“Parallel Processing using the PC-4000 RISC System,” George Nicol (Silicon Composers). George described the Novix NC4016-based, add-on boards for the PC/XT/AT, along with their PCX multitasking control and communications system which allows you to interact with each of the processors via separate
windows.

"Machine Comprehension as a Control and Planning Tool," Henry Harris (JPL). Henry, always a step ahead of the pack, gave a fascinating, but much too brief, talk on "conceptual dependency," the idea of extracting the deeper, semantic meaning from natural language. He gave just a glimmer of the potential applications of this: translation (of course), data discrimination, bandwidth conservation, and more.

"Mathcalc," David Jagelman (AT&T Bell Laboratories). David put together a fairly extensive math package which provides numerous queuing, statistical, and telephony-oriented computational facilities. This may be available from him without charge (depending on Bell Labs' policy).

"ClusterFORTH in the Factory," Adam Shakleford (FORTH, Inc.). Described FORTH, Inc.'s facility for networking IBM PCs and Z80-based, single-board computers.

"Forth and CAI From Mainframe to Micro: From Coursewriter to Forth," Brooks J. Breeden (Ohio State University). Brooks had one of the best presentations this year, describing his use of FORTH-based graphics systems for teaching aspects of landscape architecture.

"Six RePTIL Verbs and the Macintosh," Dr. Israel Urieli (College of Engineering, Ohio University). Noted that, "We have bred a community which genetically accepts Ctrl-Alt-Del as a meaningful statement."


"Object-Oriented Local Variables/Data Structures for F83," Robert H. Davis. Bob talked about a number of extensions he made to F83. These files are downloadable from the ECFB.

"LMI Forth for OS/2," Dr. Ray Duncan (Laboratory Microsystems, Inc.). Ray described and demonstrated his Forth system for the long-awaited OS/2 system.


Showed several examples of Macintosh applications that use icons for most of the user interface.

"High-Performance Networks," Dr. William Dress (Martin Marietta). Bill gave a fascinating talk about his experiments with programming a bug that learns from its activities.

"The Biological Aspects of Neural Networks," Dr. Iben Browning. Dr. Browning's talk was humorous, literate, and incisive. He discussed the potential for AI, robotics, and consciousness, combined with some scary ideas about how AIDS will impact our society. Dr. Browning was also the after-dinner speaker at the Friday evening banquet, where he expanded on the AIDS threat. A few people walked out on this talk, expressing disagreement or disbelief. In any case, this one won't go away by being ignored.

"Harris Force Processor," David Williams (Harris Semiconductor). Described FORCE (Forth Optimized RISC Computing Environment), a high-speed system suitable for real-time control systems, digital signal processing, etc. It is configurable with numerous "standard cells" (hardware modules) available from Harris. It can be used as a co-processor, stand-alone, or parallel-processing system.

"Chuck Moore's Non-linear Least Squares Fitting Routine using polypFORTH on the Silicon Composers NC4016 Board," Elizabeth Rather (FORTH, Inc.). Typical of Chuck's code, this package fits on 2.5 screens (about 25 lines of code).

"VME-based Language Processor," Tom Harkoway (Xycom). Based on Allen Winfield's MetaForth system. In describing performance, Tom defined MIPS as "Meaningless Information Provided by Salespeople."

Working Groups

Unfortunately there was only one round of concurrent working groups this year and I felt obligated to attend the one on Forth standards. Elizabeth Rather moderated, gave a recap of recent history, and described the ANSI/CBEMA process and Technical Committee membership requirements. I described the bulletin board.
board, initially funded by MCI, that will be provided for ANSI Fortth announcements, proposals, etc.

Also discussed was Guy Kelly’s plans for Fortth extension proposals to be communicated via the new FIG conference on GENie. A general discussion followed, but again there was no consensus on standards activity, though most people agreed on the need to minimize changes in a new standard.

Vendor Meeting

There were a number of concurrent vendor/user meetings. I attended Laboratory Microsystems’ meeting hosted by Ray Duncan and Mike Ham. About 20 people attended. Ray announced they would be shipping PC/Forth Release 3.2 in July. A number of enhancements were suggested. Most of these were acceptable to Ray, and he said they would try to squeeze them into the July release.

News

Look for Ray Duncan to have another bestseller next year when Microsoft Press releases his new book, Advanced OS/2 Programming, in January 1988. Let’s hope this little gold mine doesn’t totally distract him from Fortth development.

There were more people using 32-bit Fortth implementations this year than previously. The days of the 16-bitters appear numbered, other than for specialized, ROM/SBC applications.

Summary

This year we are seeing real, Fortth-based applications coming out of some advanced-technology areas: multi-processors, object-oriented programming, Lisp-like data structures, AI, and the Novix Fortth chip. Also notable is Fortth usage in some brand new areas: massively parallel processors and neural networks.

It’s fascinating to see Fortth adapt so readily to these new frontiers.

Expert systems seem to have been something of a disappointment, with few people reporting applications or new advances in this area. I wonder whether this is just a lull in activity, or have expert systems fizzled out?

This conference seemed a bit less organized than in previous years, and I was sad to learn that Thea Martin is leaving her position as editor of the Journal of Fortth Application and Research. Still, it was as enjoyable as ever, with lots of good conversation and opportunities to chat with folks I’ve previously known only by their typed output. Five days at this conference were worth a great deal in terms of information, education, and conversation.

This review was adapted from the East Coast Fortth Board (703-442-8695) with permission from the author/sysop. Jerry Shifrin works for MCI, and is one of the best sysops anywhere, if the tree can be judged by its fruit.
These are the voyages of the video game Starflight. The programming crew set about creating strange, new worlds, bravely going into greater and more realistic detail than a sci-fi, role-playing simulation — or any other game — had gone before. Michael Ham continues his special series of interviews, with Tim Lee of the imaginative — and persevering — team whose creation continues to break sales records throughout the explored universe.

MH: You work for Binary Systems, which developed a software product for Electronic Arts. Could you describe the product?

TL: It’s a role-playing game in outer space — Star Wars, where you are one of the characters.

MH: It runs on an IBM PC and takes two full disks of program and data, right?

TL: Less 40K. Yes, virtually.

MH: That’s only 10K empty per side, so that’s still pretty full. I will say that my son has been playing this game non-stop since he got it. He even maintains a log in a little spiral book to keep track of where he is and what he’s done. It’s quite a game. How many people were working on it when the product came to market?

TL: There were five members of the team. One administrative and logistics person, and one designer and three programmers.

MH: You programmed all this in Forth?

TL: And assembly language.

MH: Forth and assembly language. Now is this the original crew?

TL: Three or four members have come and gone, spending some time on the project.

MH: How long has the project been going on?

TL: Three and one-half years.

MH: When did you come on board?

TL: About six months into it.

MH: It didn’t start out quite this big in concept, did it? Did somebody say, “Let’s write a program that’ll be the biggest game ever released?”

TL: I wasn’t there at the time, but it’s my understanding from those who were that it was to be a very big-field game. But the idea of fractally generating unique planets that you can return to...

MH: Talk a little bit about fractally generated planets.

TL: It’s using the technique developed by Mandelbrot. He takes, essentially, a repeatable random sequence of numbers based on a seed. Using that, you can, in a very compact way, represent a planet with just a couple of numbers, then run it through a function and use that to determine the terrain for the surface of the planet.

MH: So each one is unique but doesn’t take up that much room, is that the advantage?

TL: That’s the advantage.

MH: David Boulton was one of the early principles.

TL: Right. He really had an impact on shaping the possibility of doing a huge environment. He wrote the first prototype fractal routines.

MH: What does the designer do?

TL: Right. He really had an impact on shaping the possibility of doing a huge environment. He wrote the first prototype fractal routines.

MH: When I played the game, I encountered these interesting alien races, and with wonderful speech patterns.

TL: That was great. The personality of
The game is Greg Johnson. He was the designer. That's what I mean. He designed the scenario; the programmers' role was more to design the game system that you could write the scenario for. You can design subsequent scenarios and not be a technical person; the programmers created a game system that could be expanded by a nonprogrammer designer.

MH: How much did those races and their characteristics evolve as you worked on it? Did they stay pretty much as he first designed them, or did you decide some of them were too evil or too simple-minded?

TL: The races were a collaborative effort — Greg Johnson, Paul Reiche III (who worked on Archon), myself, and a number of other people. We got into a sort of roundtable, but the bulk of them were done by Greg Johnson. His playfulness is manifested throughout the entire game.

MH: I don't want to go into how many alien races there are, because it's constant surprise when you are out there and you bump into someone. But did they change? You have the idea for a race. You implement it. Then you interact with them... Did you then decide, no, it's too much this or too many that?

TL: Yes, there was tuning the personality of the race. We had a data record called the race personality record, or something like that, and the designer established and modified parameters.

MH: The dialog is great. Is that also Greg's?

TL: That is also Greg. And also, again, Paul Reiche and a number of other people.

MH: Was this the first product Electronic Arts signed?

TL: No, but it was one of the early ones.

MH: It was on the first weekly status report, I recall hearing that.

TL: Yes, it's the only Electronic Arts product that has been tracked, to date, on all the different status report forms they've ever had.

MH: So it was the longest in development and, of course, the most ambitious in scope. It was released when?


MH: And sold out instantly, as I recall. And still going strong.

TL: As I understand, yes.

MH: I see notices about it on the boards, by the way. People are saying, "I can't believe it."
That's great!

MH: What aspect of the game gave you the most problems, what was the most difficult thing to accomplish?

TL: The — enclosure of the design.

MH: You set a monster in motion that didn't want to close down?

TL: Yes, that was really tough. And communicating among the programmers was also a challenge.

MH: Did you work in one place, or did each one work at home?

TL: We did a little bit of both. Bob Gonsalves, one of the other programmers, and I worked at the Binary Systems location, and Bob was getting down a couple of days a week — he was the other programmer. So that's the whole crew. Myself, Bob, Alec, Greg, and Rod on the administrative end. Rod McConnell was really the entrepreneur, the one who brought the team together.

MH: In communicating among them, how did you handle the word explosion? Large Forth projects tend to generate a lot of words. Did you maintain any sort of dictionary, or did you brief each other, did you read each other's code? How did you know what words other people had developed?

TL: We segmented things hierarchically, so one person was responsible for primitives, stack operatives, and system primitives. And everybody else used those primitives. So, of course, when you work with primitives, we'd document them (that was me, in this case) and build a glossary of graphics words. It would take a week or two of the other guys using this thing, referencing the document, for them to fully understand how the words worked. Then they'd say, "I've got a great idea for a primitive, let's put it in...." Then it would require looking at how many bytes were involved here and there. One of the more challenging things was squashing the amount of source code we have on those disks. We have over four megabytes of source.

MH: Really! Screen files or text files?

TL: Screen files, so there is a lot of white space in many places. The text is pretty dense, and with the aliens' speech is a lot of text.

MH: (grinning) And of course, the source code is contains lots of comments, right?

TL: Well, our styles vary.

MH: With really good code, you don't need comments. (laughs)

TL: (grins) That's my latest feeling. Write long definitions, and the comments and the code are the same thing. In fact, the product chronicles the development of
our programming styles. The earlier code was really terrible, and you can see the conceptual hurdles we leaped as we went on.

MH: I noticed that, too, in my programming. When I look at the early stuff, I can’t believe I wrote it. Does your program come in in overlays?

TL: Yes, there are about sixty overlays.

MH: I heard in some discussions that there is a bit of artificial intelligence in it — adaptability to the player.

TL: Yes, there are, I think, three separate expert systems in the game. That was another thing I was involved with. We wrote an expert system that allowed the game designer, who was not a programmer, to design the rules for the behavior of the aliens and communication module, how the ships behaved, how the life forms on the planets work — all of those were cooperating expert systems. He wrote the rules for them, and he didn’t have to know Forth or any programming.

MH: And that freed the programmers from much tedious work.

TL: That’s right, and it allowed him to tune the thing without needing us in the loop. It was great.

MH: What are the expert systems?

TL: There’s one for handling the aliens on the surface of a planet, one for the aliens in space — that’s the ship movement rules — and one for communications, handling the personality of the talking character.

MH: You lived with that project for a long time. Knowing what you know now, what would you do differently?

TL: I’d have to think about it a little bit to give you a real good answer.

MH: We’re satisfied with a mediocre answer at this time. You can always tune it.

TL: I think I would have recognized, based on the knowledge I’ve acquired, that we were designing a game, an entertainment product, and not a model of the universe. We spent a lot of time trying to model things that didn’t add to the play of the game. I mean Starflight, believe it or not, is a vastly cut-down subset of the original.

MH: You went for verisimilitude with a vengeance — you wanted the whole thing. In fact, you ended up with how many stars and planets?

TL: There are 810 planets and 200 star systems.

MH: And you can visit any of them in any sequence. The player is free to move around.

TL: That’s right.

MH: So you put a lot of time and effort into making ever-finier details of reality. I would think that, having a lot of people working on it, you’d get to challenging each other to get further and further into it.

TL: Yes. The whole thing acquired a personality. The morale of the team went up and down in cycles, based on a number of factors, the least of which was paying the rent.

MH: Yes, three and one-half years is a long development cycle, especially when you are paid on milestones. Just out of curiosity, are all of you science fiction fans?

TL: Yes.

MH: So you were into this as an art form already. Was your concept influenced during the course of this development, as various science fiction movies came out? This was after the first Star Wars?

TL: Oh, yes — much later.

MH: Was it influenced by anything you saw or read as you went along?

TL: A great deal. We have a rotating planet, which was why I originally hired in. The rotating planet figured prominently in Star Wars, and we had to have it. You see that view of the planet turning as you’re watching it.

MH: So this is how you got in, because they wanted the planet to turn.

TL: Yeah, and I could do that.

MH: You were the graphics guru...

TL: Graphics, virtual memory management, cache system, and object-handling data base, which was very significant in allowing the program modules to talk to each other.

MH: What is an object-handling data base?

TL: When I say “object handling,” it’s really a fancy name for a list manager, a pre-structured list manager. And I wrote a language for manipulating objects on this tree.

MH: And the object in this case would be a planet description, for instance?

TL: Planet description, alien ship... it was open-ended, in the sense that you could take an object anywhere in the universe, and then we defined a record description for that object, and we’d write code that would look at the fields within that object.

MH: What were the other programmers’ specialties?

TL: Bob Gonsalves spent a great deal of his time generating the life forms and the ecosystems for the life-bearing planets. He spent many, many months on that. He also designed the interactional planet site. The planet site is pretty much Bob Gonsalves.

MH: The planet site where you are seeking out minerals and encountering aliens.

TL: Dave Boulton’s prototypes created a
3-D terrain. I adapted that to map it onto a sphere, making it a planet. The planet builder, which places minerals, life forms, and other planetary features, was mostly Bob's work.

MH: The planet descriptions are extremely detailed regarding the ecosystem, the atmosphere, and the mass; depending on whether you've trained your science officer. If you haven't trained your science officer, the basic description of the planet is, "Don't know." But if you've got a well-trained science officer, you have an amazing amount of detail.

TL: It's hard to talk about the game in terms that someone else would understand. I took the fractal system that generated the planet and displayed the graphics. But the life-form responses — where they move around a lot, the terrain vehicle interface, all of the functions on the terrain vehicle before the player goes into outer space, that's Bob. Alec Kersco single-handedly wrote the star port module. Everything that happens in star port, he designed code for it, he wrote the code for that. Greg Johnson designed the interface. Actually, he was fundamentally responsible for the design of it. Again, good ideas came from all quarters.

MH: The star port is shopping, getting lists of things, the training, and so on. The player moves a small figure around the six stores, going through the little safari shop.

TL: The other thing Alec was responsible for was the module for communicating with aliens. I helped in the design of that, he actually had it built by another programmer who left the project, then he tore it down and rebuilt it in three weeks with the addition of the expert system.

MH: If, God forbid, all the code was lost and you redid it again from scratch, would it be much more compact?

TL: No question about it.

MH: You really learned a lot?

TL: Oh, yes.

MH: You've never really given serious thought to throwing it out and totally rewriting it...

TL: I have.

MH: Oh, you have! The perfectionist in the Forth programmer arises yet again. If one were to play this on the ideal system, what would you recommend — an AT with a composite color monitor? That's the ideal?

TL: Yes. It has a caching system that takes up about all of available RAM, so you don't even need to have a hard disk. If you have a lot of memory, it will take advantage of it.

MH: The composite monitor is for the better color. You've written an amazing system, and I'm thinking of all the techniques you developed. Do you see a way to apply them in a business application, to store and retrieve data, graphical information? You've got a way of handling data that is interesting and fast. Will you use these tools in your later applications?

TL: Yes, I don't think I will get into the business realm, but I'm certainly going to use the tools that we developed.

MH: Do you think you'll do more games, or are you interested in doing more games after this one? They're only three and a half years each, is the way I see it... in ten years you can do three games.

TL: I do have a game that I'd like to do.

MH: Do each of you have a game in mind?

TL: Yeah.

MH: What was your first encounter with Forth?

TL: My first encounter with Forth was as a contractor on a project for tax

---

Forth Interest Group Presents

Ninth Annual

FORTH NATIONAL CONVENTION

November 13-14, 1987

Red Lion Inn
2050 Gateway Place
San Jose, California 95110

FIG's 10th Anniversary Celebration

EXHIBITS

CONFERENCE PROGRAM

THE EVOLUTION OF FORTH

Past - Present - Future

The founders, writers, producers, and evaluators of Forth look at:

Forth in development
Forth at work
Forth in the future

---

Conference & Exhibit Hours
Fri. Nov. 13 12 noon - 6 pm
Sat. Nov. 14 9 am - 5 pm

Hotel Reservations
Telephone Red Lion Inn, San Jose
800 547-8010 or 408 279-0600

Registration at the door

Preregistration $20
Banquet Saturday 7 p.m.
(with keynote speaker)

Yes, I will attend the convention.

Preregistrations @ $20 $___
Banquet Tickets @ $35 $___
Total check to FIG (US funds) $___

Name __________________________
Address _________________________
Company _________________________
City _____________________________
State _______ ZIP _________
Country _________________________
Phone (____) ____________________

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

Information Phone: (408) 277-0668
preparation. It was to be written in BASIC, because at that time the only thing that was really common was BASIC. So it was going to be in BASIC, but for the work we had to do, we decided the best thing short of going to assembly language would be Forth. We used C.H. Ting's Forth interpreter written in BASIC and buried that in the product. It was fast enough for going through the form and doing calculations.

MH: Did you get into computers through a major?

TL: In high school, I read a book about computers and they had an example of a tic-tac-toe playing computer made with matchboxes. That was the very first step. But then it was about six years before I could get onto a real computer, and that was at the University of Illinois at Champaign-Urbana, the Plato system. I was in high school at the time, and a friend of mine showed it to me. I stole access time, I begged, borrowed and stole keys, I wrote a fake sign-on thing that system access, and we could then create all of our own passwords from then on in. At that time, the morality of my access didn't occur to me. Having to do it over again, I would do it differently; but nevertheless, I wrote my first games when I was about, let's see, seventeen.

MH: In BASIC?

TL: TUTOR. That's the Plato system.

MH: And what was TUTOR like?

TL: It's very loose, I don't remember much about it. It seemed like a mongrel language. BASIC-like commands. I don't think it had line numbers.

MH: When you went to college, were you going to be in computers right from the start?

TL: No, I was going to be a dentist, because my mom wanted me to be a dentist. So I went for two semesters. Well, I went for two semesters and I couldn't take the garbage. At the same time, I was working part-time at Texas Instruments in the programmable calculator division, in their software exchange. So that's where my real education took place, and when I quit going to school I just went full-time at Texas Instruments.

MH: What were you doing for TI?

TL: I was evaluating programs written for the SR56, the SR52, and the TI59 programmable calculators. People would submit programs, and we'd send them free copies of other programs in return. The programs would then be published in booklets.

MH: When was that?

TL: 1977 and '78. I was there while they were designing the TI 99/4A.

MH: That could have been a great computer.
TL: The designers knew they were screwing it up; they didn't have any choice. They were dealing with other specifications that they hadn't developed, and they said, "You know, we could make these teeny little changes, it would be great! But we can't." They had to follow this huge set of specifications.

MH: Bureaucracy rears its ugly head. How did you get to California?

TL: I came out on vacation and I just decided to stay. This is the greatest place.

MH: Was it in California that you hit Forth?

TL: Yes. Prior to that time I had never heard of it, and I'd done a lot of reading and programming, because it was a hobby. But I never heard of Forth. That would have been 1979. That was seven years since it hit the public scene.

MH: You liked it, apparently.

TL: Loved it. After seeing how it worked under BASIC, I felt a little bit ludicrous putting in these SWAPS and DROPS and shoo-bops. And then I tried to think about how I would write an interpreter, if I was going to do it; and, hey, there was no simple way. Then I got to understanding what colon definitions were, because these were just strings of things that [would go in the] interpreter and I could rule the world with this!

MH: Yeah, this is dangerous...

TL: Never look back!

MH: Was Starflight your first major Forth project?

TL: I'd done a Forth video game for Datasoft, called Genesis, and it was never shipped in the IBM format because they didn't want to ship a Forth product. It ran as fast as the assembly language versions of the Apple and the Commodore versions, and yet they didn't want to ship it.

MH: Religious differences, I take it.

TL: Well, that and they wanted to make sure it ran on the PCjr, which at that time was hot property. I decided to leave at that time and they didn't have anybody else, so it never got out.

MH: So Starflight was your first, giant project of original work—

TL: —in Forth. My first giant project was as Vice-President of Datax Corporation, a financial-planning computer-services company that did policy analysis for life insurance companies. I wrote more lines of programming, in a language called PICK-BASIC on Quan-
tel's computers, than I care to think about.

MH: Is that the small computer the agents carried into the field?

TL: No, this was a minicomputer. I think Unisys bought them. I had fifteen employees at the high point, and I managed to spend half a million dollars of someone else's money. So I've been an entrepreneurial type, and like to live on the riskier side of things than being in the corporate environment. After I got out of Texas Instruments, I decided corporations came with too much politics. If anything needed to get done, and you went by the book, it never happened. If you knew people and exchanged pull and favors, you managed to get things done. That was the way things happened.

MH: How long do you think it takes to learn Forth? I thought I knew Forth fairly early in this project, but now I realize I didn't really. What does it take to learn Forth so that your code is reasonably fluent and sound and easy to follow? It takes more than just reading Starting Forth.

TL: My own philosophy is that you need to write your own Forth.

MH: From the ground up?

TL: When you get to the point where you understand how to write your own Forth from the ground up, you are ready to write good Forth code.

MH: A demanding requirement. There is a book that takes one through that: ThreadedInterpretativeLanguages. It's for the Z80. I'd hate to do write my own Forth, though. I like to buy it out of the box.

We've covered quite a bit here. Suppose a reader decides, "That's a great idea. I'll write a computer game and make a lot of money because it's so popular, and I'll go out and be rich and famous, too." What would you say to such a person?

TL: I would say, if that's what you want to do, do it. But I tell you, the road is rocky. I don't mean in any way to dissuade someone from doing what they like to do. For me, it's been worth it, but it's been tough. As far as making lots of money, if we sell one hundred thousand units of Starflight, which is really a lot for a game — I think ultimately we will sell that many — then for three and a half years, I will have been making $6.50 an hour.

MH: Starflight, by the way, because it is so exceptional a game, may have a longer shelf life than most. You're saying, though, that if someone wants to write a game, it's interesting and they can learn lots, but they shouldn't quit their job.

TL: No. Keep your day job, that's important.

MH: If you were going to write a game, what computer would you write it for?

TL: The IBM PC.

MH: Really!

TL: Yes.

MH: Would you do a graphics game again, or would you make it text only?

TL: I think graphics are so much more appealing, as evidenced by [the fact that] all the really hot products that stay hot products, have graphical stimulators. If you can combine that with depth, then so much the better.
MH: But why the PC?

TL: There's just a lot of them out there, and it's a decent machine to develop on.

MH: The PC has enough power.

TL: It's getting cheap, too.

MH: Pretty soon, if you buy over fifty dollars worth of groceries, you get one free at your supermarket. If you are talking to someone who knows nothing about graphics, and he or she says, "Okay I want to do graphics. How do I do graphics?" Is there a book?

TL: Yes, there are several really good books. One by Newman and Sproull that's called The Principles of Interactive Computer Graphics, and another is Foley and Van Dam's, Fundamentals of Interactive Computer Graphics.

MH: Tim, thank you very much.

TL: You're welcome.

Michael Ham is a professional Forth programmer who writes extensively and works in large-scale data processing. One of Tim Lee's Starflight associates wanted us to be sure that, while giving credit all around, Tim accepts his own fair share. We hear that he not only takes a logical approach to problem solving, but applies his own inner spirit of adventure to programming.

Congratulations to the whole team: Only days ago, Starflight received Electronic Arts' awards for "Best Creation of New Worlds" and "Best Entertainment Product." Maybe the Forth really is with them!

Screen 17 is my improvement to the modification, to return an offset into the buffer, rather than the address of the matched string. This added five bytes to the code size: a PUSH to save the buffer address on the stack, a POP and a SUB in the event of a successful exit, and an additional POP in the case of an unsuccessful search (to clean up the stack before returning).

Even with my improvements, this implementation falls short of the full functionality provided by Laxen and Perry, since it is not sensitive to the case of the pattern or object string (depending on the state of the variable CAPS). Admittedly, Brother Zimmerly was designing for use in some specific, but unspecified, application of his own, where this might not be a shortcoming. However, I wish to replace the SEARCH provided in the F83 system, so it had better behave exactly as the original. I have included the code for such an assembler version of SEARCH.

This code compiles to a whopping 169 bytes — more than twice the size of the code in screen 17. I have made one concession to size by moving the conversion of characters into upper case to a labeled subroutine. In trying to arrive at a satisfactory tradeoff between size and speed, I have left the decision about converting a character with the main SEARCH code, only making the call if CAPS is on. This decision, along with a PUSH and a POP of the BX register, which I have used as a working register for this operation, could be moved to the subroutine to save additional bytes of dictionary space, or the whole routine could be embedded into SEARCH at the appropriate places, for speed at the expense of size.

While I have offered some critical observations of Brother Zimmerly's code, I wish to thank him for his effort, and Forth Dimensions for publishing it. Without this to pique my interest, I wouldn't have done this work. Being a neophyte programmer, I have learned a lot about F83 in the process, and am pleased with the results.

I enjoy the magazine and wish it came monthly.

Sincerely,
Robert Lee Hoffpauer
Richardson, Texas

Mods Quad Divides
Dear Mr. Ouverson:

The enclosed code modifies that of Robert L. Smith (FD VIII/6) for a quad-by-double divide in several respects. First, it is in a form suitable for the LMI assembler from which many Forth assemblers are derived. Second, it does the entire procedure in registers, resulting in about a 30% speed-up in the code. Third, it adds error checking by executing the divide-overflow interrupt where appropriate. Fourth, it preserves the BP register (along with SP and SI), which many 8088 Fortes use as the return pointer.

LMI's 32-bit PC/Forth+ has a similar procedure that is bulkier, a shade slower, and proprietary.

Sincerely,
Michael Barr
Mount Royal, Quebec
FIG-FORTH for the Compaq, IBM-PC, and compatibles. $35
Operates under DOS 2.0 or later, uses standard DOS files.
Full-screen editor uses 16 x 64 format.
Editor Help screen can be called up using a single keystroke.
Source included for the editor and other utilities.
Save capability allows storing Fortth with all currently defined words onto disk as a COM file.
Definitions are provided to allow beginners to use Starting Fortth as an introductory text.
Source code is available as an option, add $20.

Async Line Monitor
Use Compaq to capture, display, search, print, and save async data at 75-19.2k baud. Menu driven with extensive Help. Requires two async ports. $300

A Metacompiler on a host PC, produces a PROM for a target 6303/6803
Includes source for 6303 FIG-Forth. Application code can be Metacompiled with Fortth to produce a target application PROM $280

FIG-Forth in a 2764 PROM for the 6303 as produced by the above Metacompiler. Includes a 6 screen RAM-Disk for stand-alone operation. $45

An all CMOS processor board utilizing the 6303. Size: 3.93 x 6.75 inches.
Uses 11-25 volts at 12ma, plus current required for options. $190 $260 - $280
Up to 24kb memory: 2 kb to 16kb RAM, 8k PROM contains Fortth. Battery backup of RAM with off-board battery.
Serial port and up to 40 pins of parallel I/O.
Processor buss available at optional header to allow expanded capability via user provided interface board.

Micro Computer Applications Ltd
8 Newfield Lane
Newtown, CT 06470
203-426-8164

Foreign orders add $5 shipping and handling. Connecticut residents add sales tax.

(Continued)
PORTABLE POWER WITH MasterFORTH

Whether you program on the Macintosh, the IBM PC, an Apple II series, a CP/M system, or the Commodore, 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) .............. $80
Module relocator (with utility sources) ... $50
TAGS (Target Applic. Generation System) MasterFORTH, target compiler and relocators ........ $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

---

Hoffpauer's Screens, continued.

Scr # 15  BtTOOLS.BLK
0 | Ziemerly's search from Forth Dimensions vol IX 4 p 5 |
1 | LOADING HEX |
2 |
3 | ASSEMBLER LABEL (FINDI) DX SI MOV CX MOV BX POP POP 2FUSH FORTH |
4 |
5 | CODE SEARCH (5 sadr slen bar blen --- addr tflg junk fflig |
6 | OLD CX POP DI POP BX POP BX POP BF PUSH DX SI XCHS |
7 | CS AX MOV AX ES MOV 16 AL MOV |
8 | HERE BYTE REP SCAS 0= |
9 | IF CX PUSH SI PUSH DI PUSH DI DEC BX MOV BYTE REP CMPS 0= |
10 | IF BX POP AX POP AX POP BX DEC -1 AX MOV (FINDI) 0 JMP |
11 | THEN DI POP SI POP CX POP |
12 | ELSE AX AX XOR (FINDI) 0 JMP THEN # JMP END-CODE |
13 | LOADED DEC 15 |
14 | Code size 87 bytes including headers for SEARCH & (FINDI) |
15 |

Scr # 16  BtTOOLS.BLK
0 | Modified Ziemerly search |
1 | LOADING HEX |
2 |
3 | CODE SEARCH (5 sadr slen bar blen --- addr tflg junk fflig |
4 | CS AX MOV AX ES MOV OLD |
5 | CX POP DI POP BX POP BX POP |
6 | DX SI XCHS 0 (SII) AL MOV |
7 | HERE BYTE REP SCAS 0= |
8 | IF CX PUSH BX CX MOV SI PUSH DI PUSH DI DEC |
9 | BYTE REP CMPS 0= |
10 | IF DI SI MOV BX POP DI DEC -1 AX MOV # SP ADD |
11 | # AX MOV 2FUSH THEN |
12 | DI POP SI POP CX POP NOT # JMP |
13 | ELSE DX SI MOV AX AX XOR 2FUSH THEN END-CODE |
14 | LOADED DEC 15 |
15 | Code size 87 bytes including a header for SEARCH. |
WISC CPU/16
The stack-oriented "Writeable Instruction Set Computer" (WISC) is a new way of harmonizing
the hardware and the application program with the
opcode's semantic content. Vastly improved
throughput is the result.
Assembled and tested WISC for
IBM PC/AT/XT $1500
Wirewrap Kit WISC for IBM PC/AT/XT $900
WISC CPU/16 manual $50

MVP-FORTH
Stable - Transportable - Public Domain - Tools
You need two primary features in a software
development package...a stable operating system
and the ability to move programs easily and quickly
to a variety of computers. MVP-FORTH gives you
both these features and many extras.

MVP Books - A Series
☐ Vol. 1, All about FORTH. Glossary $25
☐ Vol. 2, MVP-FORTH Source Code. $20
☐ Vol. 3, Floating Point and Math $25
☐ Vol. 4, Expert System $15
☐ Vol. 5, File Management System $25
☐ Vol. 6, Expert Tutorial $15
☐ Vol. 7, FORTH GUIDE $20
☐ Vol. 8, MVP-FORTH PADES $50
☐ Vol. 9, Work/Kalc Manual $30

MVP-FORTH Software - A trans-
portable FORTH
☐ MVP-FORTH Programmer's Kit including
disk, documentation. Volumes 1, 2 & 7 of MVP
Series, FORTH Applications, and Starting
FORTH, IBM, Apple, Amiga, CP/M, MS-DOS,
FDP-11 and others. Specify. $175
☐ MVP-FORTH Enhancement Package
for IBM Programmer's Kit. Includes full
screen editor & MS-DOS file interface. $110
☐ MVP-FORTH Floating Point and Math
IBM, Apple, or CP/M, 8". $75
☐ MVP-LIBFORTH for IBM. Four disks of
enhancements. $25
☐ MVP-FORTH Screen editor for IBM. $15
☐ MVP-FORTH Graphics Extension for
IBM or Apple $80
☐ MVP-FORTH PADES (Professional
Application Development System)
An integrated system for customizing your
FORTH programs and applications. PADES is a
tool professional development system. Specify
Computer IBM or Apple $500
☐ MVP-FORTH MS-DOS file interface $80
☐ MVP-FORTH Floating Point Math $100
☐ MVP-FORTH Graphics Extension $80
☐ MVP-FORTH EXPERT-2 System
for learning and developing knowledge based
programs. Specify Apple, IBM, or
CP/M 8". $100

Order Numbers:
800-321-4103
(In California) 415-951-4103
FREE
CATALOG

MOUNTAIN VIEW PRESS
PO BOX X
Mountain View, CA 94040
Forth Interest Group members in good standing will elect members of its Board of Directors at the Forth National Convention on November 13-14, 1987. The Board will expand from five to seven members at that time. Robert Reiling and Martin Tracy will continue serving, and John D. Hall is running for reelection.

Kim Harris and Thea Martin are leaving the Board, after donating countless hours to the benefit of the entire Forth community. Kim is one of the five original founders of the Forth Interest Group, and has played an important role ever since then in the shaping of organizational policy and activities, and the Forth language. He will direct the program again at this year’s FORML conference.

The following individuals have been proposed by the Nominating Committee as candidates for the five vacant positions on the Board of Directors. Nomination of additional individuals is by petition, requires the signatures of 25 members in good standing of the Forth Interest Group, and must be received by September 19, 1987. A voting mechanism will be provided at the convention.

Nominees were asked to submit statements of their candidacy in 250 or fewer words:

**Wil Baden**

My first computer experience was the MANIAC at Princeton in the 1940’s, but I did not become a programmer until 1960. In the meantime I worked as a translator, private detective, editor, and other interesting but uneconomical jobs. Since then I have worked or consulted for Collins Radio, Marshall Communications, CDC, Corregated Computing Techniques, ARAMCO, Logicon, IBM, General Automation, Honeywell, Burroughs, HP, and others. At present I am “senior tool designer” at Doclz Networks, Inc., and am implementing a compiler for LUCREZIA, a programming language named after Lucretia Borgia, to enable programmers to write poisonous spaghetti code in a high-level language. Outside interests include being emcee-director of a stag show with over a thousand performances coast to coast; and lay reader, chalice-bearer, and religious instructor in the Episcopal Church. I have a wife, two daughters, two sons (one a computer hardware expert, the other a software expert, but both excellent Forth programmers), and a small assortment of dogs, cats, and personal computers. I sing second tenor.

**John D. Hall**

I appreciate being nominated once again as a director of the Forth Interest Group. I am willing to serve and I am looking forward to continued active participation.

The first five years of the Forth Interest Group showed rapid growth and many starts in several directions. The current three years have set us on a firm financial and organizational foundation. When I accepted the directorship, I set three goals to work toward. First was the consolidation and stabilization of FIG Chapters. Second was the growth of income to stabilize the foundation for future growth of the organization. Third was the consolidation and coordination of all Forth organizations into the effort of promoting Forth.

The first two of these goals have now been fully realized and, in accepting the nomination and if elected, I will continue with my third goal and I will set three other goals. First, we should broaden our reach to include professional Forth programmers and companies with products based on Forth. We can and should expect an extraordinary effort on their part in supporting the Forth Interest Group. Second, we must expand and support all educational efforts which promote Forth, whether through FORML, or programs by our members and chapters, or at universities. Third, we must provide additional communication programs to support these first two goals.

The Forth Interest Group will continue to stand as the centerpiece in the promotion of Forth.

**Dennis Ruffer**

Not many of you have heard of me, so I should explain my background. Fifteen years ago, I took a BASIC programming course and, liking it so much, I switched career directions to Computer Science. Four years later, I graduated from Western Michigan University with a major in Mathematics and a minor in Computer Science (only because they did not offer a Major in CS). I started my career in Data Processing, with RPG. Mastering that, nine years ago I was offered a job in engineering. The rest is the history of the Smart Scope, an automobile-diagnostics tool produced by the Test Products Division of the Allen Group. We now produce four products using Forth, with eight programmers and a patented diagnostic method I developed.
But why should I direct the Forth Interest Group? My goal is to increase the acceptability of Forth as a viable computer language. Forth suffers from a bad image, an image that it is unsuitable for large applications, that the principles of Computer Science do not work with Forth. I, for one, do not agree with this image, but the image is not completely unjustified. Forth is not much different than many other languages; we still need to use the tools of Software Engineering, but I have not seen much effort in that direction. Our founding fathers (of Forth) have created a very fine language, but software is more than just a language. The software life cycle includes requirements, design, testing, and maintenance, in addition to the implementation of the code. I would like to help this industry move beyond the implementation details, and on to solving the management concerns. Hopefully, I will at least make an impression.

Robert L. Smith
I have been actively involved in the Forth community for 10 years, beginning with my membership in the fig-FORTH Implementation Team. I have attended a majority of business meetings of FIG, and have been a member of the board of our local FIG Chapter. I have been the Secretary of the Forth Standards Team since 1982. I have used Forth professionally for the majority of time since learning it. I am the author of the “Standards Corner,” which appeared in Forth Dimensions, and have also published a number of other Forth papers.

If elected to the Board, I will strive to represent the various parts of the Forth community. My particular interest is in the needs of the professional Forth programmer. As a member of the Board, I would be eager to hear suggestions for improving the Forth Interest Group.

Teri Sutton
My name is Terri Sutton, and I am candidate for the FIG Board of Directors. I have been a Forth programmer for over six years. I am currently programming real-time firmware, using Forth assemblers. I have just finished a one-year term as Treasurer on the Board of Directors of the Silicon Valley FIG Chapter. I traveled to Taiwan and China last fall to talk about the history of FIG. These experiences have led me to the conclusion that FIG isn’t sure what its role should be. As a board member, I will work to determine clear goals for the organization and how to implement them. For example, I want FIG to be able to support its members better by offering workshops and seminars. I would like to look into improving the employment service now offered. Forth isn’t getting the credit it deserves, and companies that could profit from using Forth are choosing not to use it. I think FIG should try to change this, possibly by educating programmers and/or companies. I believe that, as a Board member, I can improve FIG’s future.
U.S.A.

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

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

• ARIZONA
  Phoenix Chapter
  Dennis L. Wilson (602) 956-7578
  Tucson Chapter
  John C. Mead
  2030 E. Broadway
  Jungkind
  12700 S. 4th & Main
  Noshir
  & Dimensions

• CALIFORNIA
  Los Angeles Chapter
  Hawthorne Public Library
  12700 S. Grevellea Ave.
  Phillip Wasson (213) 649-1428
  Monterey/Sanluisa Chapter
  Bud Devins (408) 633-3253
  Orange County Chapter
  Fullerton Savings
  Huntington Beach
  Noshir Jesung (714) 842-3032
  Sacramento Chapter
  4th Wed., 7 p.m.
  Tom Gormley (916) 444-7775
  San Diego Chapter
  Thursdays, 12 noon
  Guy M. Kelly (619) 454-1307
  Silicon Valley Chapter
  4th Sat., 10 a.m.
  H-P, Cupertino
  George Shaw (415) 276-5953
  Stockton Chapter
  Doug Dillon (209) 931-2448

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

• OHIO
  Akron Chapter
  3rd Mon., 7 p.m.
  McDowell Library
  Thomas Franks (216) 336-3167
  Athens Chapter
  Isreal Lordi (614) 594-3731
  Cleveland Chapter
  4th Tues., 7 p.m.
  Chagrin Falls Library
  Gary Bergstrom (216) 247-2492
  Dayton Chapter
  2nd Tues. & 4th Wed., 6:30 p.m.
  CPC. 11 W. Monument Ave., #612
  Gary Ganger (513) 849-1483

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

• OREGON
  Greater Oregon Chapter
  Beaverton
2nd Sat., 1 p.m.
Tektronix Industrial Park, Bldg. 50
Tom Almy (503) 692-2811
Williamette Valley Chapter
4th Tues., 7 p.m.
Linn-Benton Comm. College
Pamm McCaug (503) 752-5113

• PENNSYLVANIA
Philadelphia Chapter
4th Sat., 9 a.m.
Drexel University, Stratton Hall
Melanie Hoag (215) 895-2628

• TENNESSEE
East Tennessee Chapter
Oak Ridge
2nd Tues., 7:30 p.m.
800 Oak Ridge Tunkpike,
Richard Secrist (615) 483-7242

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

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

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

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

• WISCONSIN
Lake Superior FIG Chapter
2nd Fri., 7:30 p.m.
Main 185, UW-Superior
Allen Anway (715) 394-8360

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

Milwaukee Area Chapter
Donald Kimes (414) 377-0708

INTERNATIONAL

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

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

• BELGIUM
Belgique Chapter
4th Wed., 20:00h
Contact Luk Van Loock
Larksdreft 20
2120 Schoten
03/658-6343

Southern Belgium Chapter
Contact Jean-Marc Bertinchamps
Rue N. Monnom, 2
3176 Waregem
071/213858

• CANADA
Northern Alberta Chapter
4th Sat., 1 p.m.
N. Alta. Inst. of Tech.
Tony Van Myden (403) 962-2203

Nova Scotia Chapter
Halifax
Howard Harwitz (902) 477-3665

Southern Ontario Chapter
Quarterly, 1st Sat., 2 p.m.
Genl. Sci. Bldg., Rm. 212
McMaster University
Dr. N. Soltesse (416) 525-9140
ext. 3

Toronto Chapter
Contact John Clark Smith
P.O. Box 230, Station H
Toronto, ON M4C 5J2

Vancouver Chapter
Don Vanderweele (604) 941-4073

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

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

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

• FRANCE
French Language Chapter
Contact Jean-Daniel Dodin
77 Rue du Capucins
69003 Lyon 03
04/38-67-77

FIG des Alpes Chapter
Annely
Georges Seibel, 50 57 0280

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

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

• IRELAND
Irish Chapter
Contact Hugh Dobbs
Newton School
Waterford
051/78757 or 051/74124

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

• JAPAN
Japan Chapter
Contact Toshi Isobe
Dept. of Mineral Dev. Eng.
University of Tokyo
7-3-1 Hongo, Bunkyo 113
812-711 ext. 7073

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

Volume IX, Number 2

43
Forth Dimensions
NOW AVAILABLE

"For all versions of FORTH, with additional information on extension wordsets of the MMFSFORTH System."

FORTH:
A TEXT AND REFERENCE

Mahlon G. Kelly
Nicholas Spies

PRENTICE HALL SOFTWARE SERIES

FORTH INTEREST GROUP

$22 EACH

FORTH INTEREST GROUP

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