Forth Resources via Modem

XMODEM Tutorial
On-Line Documentation
Forth Source Formatter
Dual-CFA Definitions
THEY'RE HERE!

at

SOFTWARE COMPOSERS

Delta Development System, Model 1
* Delta Board, seven socket backplane, 56K-word CMOS memory board, power/battery card, serial cable, portable Delta Box and cmForth.
* Fully assembled, tested, and ready to use with 90-day warranty.
* User manual and access to 24-hour user bulletin board support.

SCFORTH
* ForthB3 Standard-like word set with a utility module.
DELTAC
* Extended Small-C programming language for the Delta Board.
* On PC translates C to Forth for upload and operation on Delta Board.

The Delta Evaluation System $895
* 4 MHz Delta Board with Novix NC4000 Forth Chip on board.
* cmFORTH programming language interpreter and compiler in EPROM.
* User manual, board schematic, and user bulletin board support.
* 4K 16 bit words of static RAM and 4K words of EPROM.
* 8 selectable 256 word data stacks and return stacks for multi-tasking.
* 21 independently programmable single bit I/O ports.
* 4 1/2" x 6 1/2" board with 72-pin edge-connector bus with all major Novix signals.
* Delta Regulator Base with attached connector and single 5-volt wall mount power supply.
* Reset switch and serial port on board with RS232 connector and cable.
* 90 day warranty.
* Fully assembled, tested, and ready to use.

Additional Novix chips available for $195, 1-9 quantity.

Software Composers is an authorized Novix Distributor

SC-1000CPU Users Manual: #75 shipping/handling included in U.S. Covers Delta Board, Novix chip, and cmForth. Deductible from price of Delta Board in event of purchase. PC clone terminal/disk server F83 software provided. Six month subscription to 24 hour user bulletin board.

"I'm delighted to see Software Composers' board on the market. It provides incredible capability and versatility with minimal parts, size and price. An excellent introduction to the new generation of hardware and software."

Chuck Moore, November 1985

COMING SOON!

SC-100OPBB Power/Battery Board
* 4.5"x6.5": compatible with Backplane and Evaluation System power supply.
* Uses 4 D-size NiCd batteries with recharging circuitry.

SC-100OPER Peripheral Board

For product data and ordering information, write:

SOFTWARE COMPOSERS

Software Composers
210 California Avenue #F
Palo Alto, CA 94306
(415) 322-8763
FEATURES

9 XMODEM Tutorial
by John S. James
The public-domain XMODEM protocol has long been in widespread use for error-free data transfer among personal computers. Because of the various states in the execution of the program and their associated timeouts, it may not be obvious at first how to organize the algorithm gracefully in a structured language. This implementation is a tutorial on Forth programming and is also a useful, complete application.

21 On-Line Documentation
by John J. Wavrik
The best form of documentation for a Forth application is the source code. F83's VIEW or LOCATE show the screen on which a word has been defined. Here is the same facility implemented for fig-FORTH, Kitt Peak VAX-Forth, MMS-Forth and MVP-FORTH. A great tool!

25 Forth Resources via Modem
by Gary Smith
More Forth material than ever before is available on private and vendor-supported electronic bulletin boards and on commercial information systems. FIG members with modems and communications software have more opportunities than ever before to expand their knowledge and their personal dialog with other Forth experts. Here's where to find them!

27 Forth Source Formatter
by John Konopka
The problem of neatly formatting source code is accentuated for Forth programmers, who often must add code to the fixed size of one screen. The program presented here will take the most compact, confusing source code as input and will turn out a cleanly formatted listing with new lines for every colon definition, proper indenting for structured constructs and more.

30 Dual-CFA Definitions
by Mike Elola
Decomposing functions is a critical part of Forth programming. Dual-CFA definitions provide the benefits of decomposed functions in situations where decomposition would not be possible normally. By first exploring more conventional Forth programming techniques, such as vectored execution, the author discusses various issues that showcase the advantages of dual-CFA words.

DEPARTMENTS

5 Letters
33Advertisers Index
34 FIG Chapters
Mach1™

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

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

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

---

**Mach1 on the Mac features:**

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

68020 compatible

*Comes with Switcher/Edit and 400 page manual*

---

**Mach1 on the ST features:**

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

68020 compatible

*Comes with 300 page manual*

---

**Mach1 in EPROM:**

- 16K FORTH Kernel
- 16K 68000 Assembler
- 16K Disassembler/Debugger

*(Available immediately on Gespac G-64 68000 board)*

**Call for source licensing arrangements...**

Executes sieve on 16MHz 68020 in 1.3 seconds

---

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

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

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

---
Saddled With Benchmarks

Dear Mr. Ouverson:

Now that Mach1 (our Forth compiler) is becoming a rather efficient development system, we are somewhat concerned with the benchmarks that Forths are saddled with when comparing them to other languages. The only sieves we've seen people use must be corrupt versions of those called “BYTE” and “Colburn” sieves. They are not only wrong (there are 1028 primes between zero and 8192, not 1899), but judging from the stack manipulations, were perhaps rewritten by people who were unfamiliar with, or unfriendly to Forth. Has someone pirated our benchmarks?

When a person truly wants to find prime numbers (and, indeed, our customers are always asking us for the higher primes), he'd like to do it simply, efficiently and correctly. Could you please publish the correct “BYTE” and “Colburn” sieves, and this Forth-83 sieve as ones which do just that?

Thank you,

Terry Noyes
Palo Alto Shipping Co.
Menlo Park, California

DECIMAL
8192 CONSTANT size
size SORT 1+ CONSTANT flicklimit (if you don't have SORT, just use 4 / to be safe)
VARIABLE flags size UVALLOT

: primes (--number of primes) ( does the primes once)
   flags size 01 FILL
   ( initialize the array )
   0    ( prime counter )
   size 2
   ( range of numbers to check )
   DO
   flags I + @
   ( see if I is a prime )
   IF
   I flicklimit <
   ( no need to try and flick flags once you get past size*0.5, just keep adding up the primes )
   IF
   I ( this is a prime. Used to increment the +loop )
   flags size + flags I + I +
   ( range of addresses to tag )
   DO
   0 I C) DUP ( flick flags at multiples of I )
   +LOOP
   DROP ( drop the I that was used for +LOOP )
   THEN
   I+ ( increment the prime counter )
   THEN
   LOOP ;

How it works: Initialize an array to all ones. Start with the first prime number, which is two. Clear all bytes in the flags array that are multiples of two. Then do it for the threes. The fourth byte in “flags” is a zero, so skip to five. Since four is non-prime, all of its multiples were handled by another number, two in this case. You only have to continue this process up to the integer square root of the array size, since in this case 91*91 is not in the array and the 91*90 spot was already zeroed by a previous prime number. If you dump the array, you'll see that bytes 2, 3, 5, 7, 11, 13, etc., are marked with ones, and the others have been zeroed. (While this program runs, it counts all the prime numbers it finds.)

Public-Domain Floating Point and Double/Quad Precision

Dear Marlin,

Forth has gone too long without a complete, standard wordset for extended arithmetic functions! Instead of continually reinventing the wheel with hardware-dependent math packages and spending time writing yet another version of \texttt{D*}, the Forth community needs to expend its energy addressing new horizons.

In order to foster the rapid growth of a standard, I am placing my book, \textit{MVP-FORTH Integer and Floating-Point Math} (MVP book series, volume three), in the public domain. This book contains a complete, machine-independent, high-level MVP-FORTH glossary and implementation for thirty-two-bit integer math, sixty-four-bit integer math and thirty-two-bit floating-point math with transcendental functions. The book also includes assembler source code for critical words on popular CPUs, to give execution speeds comparable to other high-level languages.

The math package included with the book has been stable and in active use for several years. The word definitions are similar or identical to many of the other proposed "standards." However, this wordset has the advantages of machine independence and public-domain implementation. The book and source code on disk may be ordered from Mountain View Press.

I think that a standard for Forth arithmetic will only emerge from the evolution of a public-domain implementation in common use. I encourage anyone with comments on, or improvements to the implementation described in my book, to write me in care of Mountain View Press.

Phil Koopman
North Kingstown, Rhode Island

Dictionary Magic

Dear Marlin:

Here is a bit of magic for those who need to generate more dictionary space without reducing the number of options they are loading.

The method requires that there be some space available elsewhere, large enough to be useful. I use what would otherwise be a
Dear Mr. Ouverson,

In writing words to read the clock on my AST board from F83, I ran into all the problems Laughing Water referred to in his letter, “You Screen, I Scream” (VII/2). Adding definitions to the existing code immediately filled up a screen, then what do you do? Also, keeping shadow screens lined up is difficult.

In good FORTH programming practice, each word should do one job and do it well. Then the next higher level word does not have to concern itself with the details of how the lower-level word works. This principle of hiding complexity is fundamental to modular programming.

Then there is the FORTH editor! It hides nothing. The programmer must keep track of screen numbers, line numbers, line length and a host of other items. If a screen is full and an additional definition is needed, the screens must be moved manually. Any reasonable text editor will insert text anywhere in a file, moving the rest down to make room. Also, my screen is eighty columns wide and it is all too easy to enter a definition that is sixty-five characters long and lose the last word.

I propose that the editor and the loading of definitions be modified to hide the 1K screen limit. The FORTH editor should work like a standard text editor. Comments and definitions can be entered anywhere in the file. To maintain compatibility with block-oriented systems such as polyFORTH, each 1K block could be treated like a page. User variables could be defined that would limit the maximum range the editor could use, to avoid overwriting the wrong blocks. In a system that runs under a host operating system, such as F83 under DOS, these “pages” or screen numbers would maintain compatibility with the VIEW command. The programmer does not have to keep track of these screen numbers while programming.

Since this is a major change to the current FORTH editor philosophy, it may not be acceptable. May I suggest that at the minimum, an editor should be free format, lines terminated with CR-LF and not displayed with line numbers. Even if restrained to the 1K limit, this would be much better than the standard FORTH editors. If implemented with a multi-line move command with a large enough buffer to hold a definition, it would be a great improvement. Then a few words would automate the screen copy or convey, and would move screens as required to make space for new definitions.

If any FD reader has a program to read and compile F83 FORTH programs from a DOS text file for the IBM-PC, please write to me. I would also like to know if anyone has implemented a FORTH editor that uses the IBM-PC cursor keys and produces a straight ASCII text file or the screens as I have outlined above.

Sincerely,

Ramer W. Streed
North Mankato, Minnesota
An invitation to attend the eighth annual

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

Following Thanksgiving
November 28 - 30, 1986

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

Theme: Extending Forth towards the 87-Standard
FORML isn't part of the Standards Team, but the conference is an opportunity to present your ideas for additions to the Forth standard. Papers are also welcome on other Forth topics. Meet other Forth professionals and learn about the state of the art in Forth applications, techniques, and directions.

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

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

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

Registration and abstracts are due September 1, and final papers are due October 1, 1986. Make your reservation now and get your registration packet. Space is limited, advance registration is required.
Conditional Teaching

Dear Marlin,

This letter is inspired by Ronald Apra's article on teaching Forth. I've been teaching Forth for several years. My situation is different from Mr. Apra's: I'm teaching university students who already know some other language. Several years ago, however, my wife and I worked with some elementary school students in BASIC, and she has also taught BASIC and Pascal to adults.

Conditional structures, no matter what the language, create a problem for beginners. A student's first language is a barrier. Teaching is often inefficient because we are trying to give students the answers to questions they have never asked. The question in the students' minds is, "Why are we doing this?" My wife suggested to a group of upper elementary students that they write programs to do something they would like to do. One girl wanted an "address book." One boy wanted to write a video tennis game (it was suggested that he first try to get a "ball" to bounce back and forth on the screen). Both of these students accomplished their goals. In the process, they had to deal with "How can we do this?" and, as you'd expect, they both came to understand looping and conditional structures quite well. The teaching of computer programming at all levels would benefit by the production of a collection of interesting tasks that develop a need for certain constructs.

"Keep it simple" is indeed a hallmark of Forth. Forth is perhaps the only language whose implementation is simple enough that the language can be learned semantically instead of syntactically. The major unifying principle is "action": each word has an action which it performs when the word is executed. This unifying principle explains why it is desirable that arguments on which the word acts be available before it executes (i.e., reverse Polish syntax). The conditionals can be understood in terms of their action during compilation. A decompiler is an important tool at this stage. If students already know why an IF . . . THEN construct is needed, and they see how IF compiles a conditional branch and THEN resolves the displacement, they automatically understand the Forth syntax. fig-FORTH, it should be noted, is advantageous for this approach because system words (like OBRANCH) are named and full word names are stored (facilitating decompilation). A student who learns Forth in this way develops a mental image of what each word does — and this eliminates the need to memorize syntax rules.

I repeat again that I'm working with university students with prior programming experience. I'm not sure how far "down" this approach would extend. I am sure, however, that the conceptual simplicity which Forth offers is an important component of its power. I do see potential in

(Letters continued on page 33)
The public-domain XMODEM (MODEM?) protocol has come into widespread use for error-free data transfer among personal computers. It has other commercial uses besides, although it is less than ideal for communicating with mainframes over packet-switching networks, a task for which it was not designed. We are publishing this implementation as a tutorial on Forth programming, and also for its practical usefulness. It is not the only XMODEM implementation published in Forth (see references below).

In addition, this two-part article will touch on the philosophy of designing software modules for effective use in libraries. We will also look at compatibility among Forth systems, and at the marketing issues raised by the question of why this author uses the public-domain F83 implementation as a publication language, while using proprietary systems purchased from others for many software-development projects.

Overview

Implementing XMODEM is not as easy as might be assumed. Because of the various states in the execution of the program and their associated timeouts, it may not be obvious at first how to organize the algorithm gracefully in a structured language.

Concerning timeouts, the program should be less strict than the traditional XMODEM when it is used for communicating with mainframes (e.g., services like CompuServe, which supports XMODEM). Since these longer timeout values probably will not interfere with micro-to-micro communication, except to cause some additional delay in recovering from certain errors, we have used them here, with the original XMODEM timeouts in comments.

One challenge for this particular implementation was to make the CRC option efficient in high level. CRC (cyclic redundancy code) was not part of the original XMODEM protocol, but was added later for improved error control. The problem for us is that the CRC algorithm involves much bit shifting, and standard Forth doesn't include efficient shifting. Ordinarily, one would write a code word to do the shifting, but for this example we want to avoid using code in order to run on different computers. So we used some programming tricks, explained below, to implement the CRC efficiently.

This software accumulates the CRC while the characters are coming in, so it must keep up with the characters. But note that the program could be designed to not have to keep up: XMODEM always sends blocks with 128 bytes of data, and reasonable delays between the blocks are allowed. If you have a very slow computer and a very fast modem, you might need to change this program to simply store the characters as they come in, then compute the error check later. The price would be a small but noticeable slowdown in receiving files.

One more challenge in implementing XMODEM is to make it work gracefully with the different variations which exist. Some existing implementations do not talk to some others, mainly due to the timeout differences mentioned above.

What is XMODEM?

XMODEM was developed by Ward Christensen, and has been extended by others. It has been used on bulletin boards since around 1980.

In XMODEM, all data is transmitted in 128-byte blocks. Each block also contains a few bytes of header and error-test information. The receiving program must test each block, and send either an ACK (acknowledge, ASCII 06) to indicate a correct block received, or a NAK (negative acknowledge, ASCII 21 hex) to indicate error. If it sends the NAK, the sending program will retransmit the block.

Each block contains:
- A start-of-header byte, ASCII 01.
- The record number mod 256, starting at 1.
- The ones complement of the record number.
- 128 bytes of data.
- CRC high and low bytes (or a one-byte checksum, if CRC is not used).

The conversation between the computers is receiver driven. This means the sender doesn't do anything until asked by the receiver (except to time out eventually, if the receiver doesn't work properly).

To start the process, both computers must be made ready, one to send and the other to receive. At first, nothing happens. Then the receiver times out, because it failed to get any record; when it times out, it sends a NAK. The NAK tells the sender to retransmit — in this case, to transmit the first record. The receiver ACKs or NAKs it; the transfer process has now started. (If CRC is used, the receiver must send a C the first time — ASCII 43 hex — instead of NAK, in order to tell the sender to use CRC. In either case, the receiver doesn't actually have to wait for the first timeout, but can send the C or the NAK earlier. Some programs try C a few times, and if the sender doesn't respond, they try NAK, in case the sender doesn't support CRC.)

After the file has been transmitted, the sender sends a single character, EOT (end of transmission, ASCII 04). If the receiver gets it, it sends an ACK, and the file transfers are finished.

Unusual Situations

If a NAK gets garbled, the sender won't do anything — until the receiver times out and sends another NAK.

Sometimes an ACK gets garbled. Then the sender will re-transmit a valid record. The receiver, which always tests the record number, throws away the extra copy.

The sender should not treat any non-ACK as a NAK. Especially it should not treat control-S or control-Q, which the receiver may send to ask for a pause in transmission, in this way. If the sender does honor control-S and control-Q (not strictly necessary, since retransmission can take care of most problems), the control-Q to restart transmission may occasionally get lost. Both sides can be designed to handle this case, even if the other side would wait forever, but we haven't done so in this tutorial.

If the record number on a received block is neither the expected one nor a repeat of the previous one, a fatal synchronization error has occurred, and transmission must be terminated. A transmission error in the
FEATURES

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

COST

130 page manual - $30.00
8K EPROM with manual - $100.00

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

Bryte Computers, Inc.
P.O. Box 46, Augusta, ME 04330
(207) 547-3218
block number will not cause this situation, because the header number and its ones complement must correspond; if they don't, the record will be discarded immediately, before being checked for the fatal error.

Timeouts and retries must keep either computer from hanging indefinitely, no matter what the other does or doesn't do. XMODEM specifies that the timeout between blocks is ten seconds, and within a block the timeout between characters is one second. (We use twenty seconds and ten seconds respectively, as recommended for CompuServe, to assist communication with busy mainframes over packet-switched networks, where delays of over a second are common.) If an error occurs, it should be retried for a total of ten times before the program gives up.

Using the Code

Part I of this two-part article implements the heart of the process. This code lets you receive a record, using the CRC. (The code presented here also includes a dumb-terminal program, named FT, to let you log on to the remote computer and do whatever is necessary to tell it to start sending a file using XMODEM.)

Before describing how the code works, we'll explain how to use it.

(1) Make sure it compiles. The code runs on the public-domain F83 implementation of the Forth-83 Standard; and screen 1 contains some simple name changes to adapt it to PC/Forth, a Forth-83 Standard implementation from Laboratory Microsystems, Inc. These changes in screen 1 may apply to other Forth-83 Standard systems also.

If your system has a different assembler, the word CALL-serial may need to be rewritten; or perhaps your system already has such a word.

(2) If you have a different microprocessor, or have one of the few semi-PC-compatible which don't support the BIOS calls, you will have to rewrite MINITIALIZE, MREAD and MWRITE for your equipment. (Note that these words actually refer to the port, not the modem.) Your modem too may need some initializing — for example, a dial command and the phone number to dial — but with most modems you can provide these manually by simply typing the characters after you have executed the dumb-terminal program and are talking to the modem. We didn't need to build the modem's initialization into the program, in this example — fortunately, since there are many incompatible modems.

You might want to change the speed from 300 to 1200, in screen 5.

(3) When you get a good compile, execute FT, the dumb-terminal program. Now you are talking to the modem.

If the system seems to have crashed, it's probably because the modem isn't echoing your characters or saying anything else back to you. You might type a control-Z, which exits from the dumb-terminal program, to make sure you can get back to Forth in order to check that the program hasn't really crashed.

Get back into the terminal mode, and use whatever character or sequence your modem needs to get its attention and into command mode. Usually it's possible to talk to the modem and get something back from it — an "OK," a status report or whatever — before even connecting the telephone. This way you can make sure that everything is working so far, before introducing the added complexity of the remote computer.

(4) Now use the terminal program to talk to a remote computer. Most modems these days are auto-dial — you can type in the phone number as part of a command. In either case, follow the instructions for your particular modem (often easier said than done).

(5) To test the XMODEM record-receive software, call up a bulletin board or a service which supports XMODEM. Do whatever commands are necessary to tell it to start sending a file with XMODEM. Note that for testing this code (in Part I of this article), you need to call a system which can support the CRC error check. (Part II will include code for the simple checksum, also.)

If you use CompuServe or a similar system for the test, make sure to ask for a "text," not "binary" file. Many mainframes have a thirty-six-bit word and normally store text in seven bits per character; so if they must store eight bits, as when saving object programs for micros, they use a special translation such as Intel hex format, which will look like garbage if you receive it.

Once the remote program is in XMODEM and waiting to send data, you have a reasonable timeout period, probably at least a minute, to ask it to start. Get out of the dumb-terminal program and back to Forth with the control-Z. You're still online with the remote computer, and can execute FT again later to get back into the terminal mode.
To start the remote system, send the letter C (ASCII 67 decimal). Since the data may start coming quickly, the command to receive it should be on the same line, as in

67 MWRITE XGET

XGET is a short test word defined on screen 12. It prints the number of characters read (should be 133), and a flag (0 = good, 1 = end-of-file, 2 = an error such as too few characters or bad CRC).

The record transmission will take a little over one second at 1200 bps, four seconds at 300. If everything worked, XGET will print 133 and zero. Dump some bytes from DATA to see if you got the start of header (01), the record number (01), its one complement (FE hex) and the data. Or just use DATA 3 + 128 TYPE to see the text. (Some Forth systems will TYPE carriage returns and line feeds as control characters, others will perform them instead.)

If you got 133 bytes but an error, it is possible that the software is working properly and XMODEM detected a transmission error, as it should. If it seemed that nothing happened, wait half a minute or so for any possible timeout, and then check the results from XGET. If one or two characters are missing, there may have been transmission errors. If no characters were received, perhaps the remote system does not support CRC; try

<NAK> XWRITE XGET

and see if 132 characters are received; there will be timeout and an error because the code given here does not support the simple checksum, only the CRC.

To try for another record, use

ACK XGET to get the next one, or

NAK XGET for retransmission of the last record received.

There is another way to check what has happened. The dumb-terminal program keeps a record of your session — all characters sent both ways — at TSAVE. If things don’t work, you can dump that data.

(6) To end the test, it’s helpful to get back to the terminal mode and log off the remote system. But if you haven’t reached the end of file, it may be hard to get the sending program to abort the transmission. If nothing else will do, unplug the phone.

How the Code Works

Screens 2 and 3 provide the low-level I/O — MINITIALIZE, MREAD and MWRITE — to initialize, and to read and write characters. As explained above, you may need to rewrite these words.

Note that MREAD accepts a timeout value in milliseconds. You may want to adjust the delay loop in screen 4 to make the values relatively accurate for your computer. There are better ways to handle timing, of course, but we kept it simple for this tutorial.

Screen 4 defines words to save a record of the session — a copy of all I/O — in a memory area which can be examined later to trace problems.

Screen 4 illustrates that comparisons of addresses must be unsigned — using the Forth–83 Standard word u<. Otherwise, the result would be correct only if both addresses were on the same side of the 32K boundary (in a 64K system). A program could work perfectly until a recompilation just happened to cause a buffer to span that boundary. So instead of using MIN to get a minimum, as part of the process of preventing the trace of the session from overflowing its save area, we defined UMIN, an unsigned minimum, and used it instead.

Screen 5 implements a dumb-terminal program, T. We use a short name to make it easy to get into and out of terminal mode (control-Z gets out of the terminal mode and back to Forth). T includes the machinery to save the record of a session — all characters sent either way. Note that SAVE-INIT is executed at compile time; otherwise it would be easy to forget when testing. In a product, T would execute SAVE-INIT; but here that wouldn’t be appropriate, because during testing you should be able to get into and out of terminal mode without wiping out the record of the session so far.

Screen 5 also redefines MREAD and MWRITE, to create new versions which save a copy of the data they transfer. Note that we could not use these redefinitions in T, because T is constantly doing MREAD which times out; it would fill the session-save area with garbage.

Screen 6 sets up constants and variables for the XMODEM code itself.

Screen 7 does the \"line purge\" — waiting until the sender stops transmitting — which XMODEM requires before a NAK, to avoid possible confusion.

Screens 8 and 9 are the hard part — computing the CRC, and doing so efficiently in high level.

Here is the algorithm. Start with a 16-bit accumulator set to zero. Take the 128 data
Scr # 5  A:XMODEM.
0 \ Dumb terminal, for test. 'T' to run, control-Z exit 2-17-86JJ
1 : T \ -- Dumb terminal program
2 300 INITIALIZE \ 300, 1200, or other
3 FALSE \ Loop control - set to TRUE to exit
4 BEGIN
5 0 MREAD \ No need for timeout delay here
6 DUP -1 = IF DROP ELSE 127 AND DUP SAVE EMIT THEN
7 KEY? IF \ If key typed, send it, unless control-Z
8 KEY DUP 26 = IF DROP 1 \ Exit
9 ELSE DUP SAVE MWRITE THEN
10 THEN
11 DUP UNTIL DROP ;
12 \ Apply SAVE to later I/O - can't use this in T, due to timeout
13 MREAD MREAD DUP SAVE ;
14 : MWRITE DUP SAVE MWRITE ;

Scr # 6  A:XMODEM.
0 \ XMODEM variables and constants 2-17-86JJ
1 VARIABLE RECORD \ Address of 133-byte area
2 VARIABLE NCHAR \ Number of characters transmitted
3 VARIABLE XTIMEOUT \ Timeout value for read, in milliseconds
4 VARIABLE DATA-BYTE \ Used in CRC
5 1 CONSTANT <SOH>
6 4 CONSTANT <EOT>
7 6 CONSTANT <ACK>
8 21 CONSTANT <NAK>
9 10000 CONSTANT CTIMEOUT \ 1 sec, timeout between characters
10 20000 CONSTANT BTIMEOUT \ 20 sec, timeout between blocks
11 \ In traditional XMODEM, above timeouts are 1 sec and 10 sec
12 1000 CONSTANT PTIMEOUT \ Try 1 second to purge the line
13
14
15

Scr # 7  A:XMODEM.
0 \ Purge the line (wait for transmission to stop) 2-17-86JJ
1 : XPURGE \ -- Wait till transmission stops
2 BEGIN
3 PTIMEOUT MREAD \ Wait till no more characters
4 -1 = UNTIL ;
5 6
7 8

Scr # 8  A:XMODEM.
0 \ Compute CRC 2-17-86JJ
1 HEX
2 : 1BIT \ xaccu1 mask -- xaccu2 Apply 1 bit of DATA-BYTE
3 SWAP ( Save mask ) DUP 8000 AND IF \ Bit to shift out
4 DUP + \ Shift left
5 SWAP DATA-BYTE @ AND ( Apply mask ) IF 1+ THEN "Shift"
6 1201 XOR \ CRC polynomial
7 ELSE \ Same thing, only don't do the CRC
8 DUP +
9 SWAP DATA-BYTE @ AND IF 1+ THEN
10 THEN ;
11 12
13
14
15 DECIMAL
bytes, and shift each bit (high bit first) of each byte into the accumulator. Throw away anything shifted out of the accumulator — but whenever a 1 bit is shifted out, exclusive-or the accumulator with the magic number 1021 hex (a number specified in the CCITT CRC-16 standard).

We will do this with the 128 data bytes as they come in. The algorithm also requires two zero bytes to be put through at the end. The final value in the accumulator should match the CRC bytes which were transmitted.

How do we do all this efficiently without an efficient shift — except a left shift by adding a number to itself, which doesn't even keep track of anything shifted out?

This program keeps the accumulator on the stack. UPDATE-CRC, on screen 9, takes advantage of the fact that the data byte need not be physically shifted, if eight different masks are applied to see if there is a bit in the position from which it would have been shifted out. If so, a bit is simply added to the accumulator (after the accumulator has been "shifted" by adding to it a copy of itself). The 1+ in IBIT causes the effect of shifting a 1 into the accumulator.

To get around the fact that + doesn't keep track of overflow, simply test the high bit of the accumulator with 8000 AND (hex) before adding to itself.

The result of this test needs to be used later — after the rest of the work on the accumulator. To save a little time, IBIT doesn't store that result, but instead has parallel code sequences — in effect, "storing" the bit shifted out in the control of the program itself.

In screen 9, GET-CRC puts only the 128 data bytes (of the 133 bytes transmitted) and the two required zeros into the CRC. (We don't actually use GET-CRC here, since we accumulate the CRC while the characters are being received; but the word can be handy for testing.)

The rest of the code is fairly straightforward. XREAD reads n characters, or until
timeout. XCHECK tests the data received. XGET, ACK and NAK are convenient test words to run from the terminal.

Notes on the Code

The code example will not work with some external modems due to RS-232 incompatibilities. MREAD and MWRITE will timeout in one second and not transfer any data. You may be able to avoid the problem by setting switches on the modem, or by using a special cable.

Or you could patch MREAD and MWRITE to bypass the operating system and access the port directly. Replace the definition of MWRITE with:

```french
3F8 PCI
```

where PCI writes one byte to a port. In MREAD, replace 200 CALL-PORT1 with 3F8 PCI. (Some systems may need one of these patches without the other, or may need a more elaborate MREAD.)

Also note that the crude dumb terminal program may lose characters between lines, while the screen is scrolling. XMODEM is not affected, however; the dumb terminal is used only to navigate through the remote system to get its XMODEM started. We chose not to complicate the article with the testing.

Discussion — Forth Libraries

I am implementing a Forth library system to allow developers to use off-the-shelf modules (see “Forth Component Libraries,” reference below). The code here does not use the library facilities, since they are not yet available; but this code was designed to be adapted and used in the library.

The library will hide the internals. It will show the XMODEM words to the outside world on three different levels. The low level will be MINITIALIZE, MREAD and MWRITE — not really XMODEM at all. The middle level will have words like XGET which read and write records. The highest level will receive and send whole files, creating, opening and closing them, etc.

Note:

(1) The low-level words don’t take a full set of all possible transmission options. For example, MINITIALIZE takes advantage of the fact that XMODEM specifies no parity, eight data bits and one stop bit; usually these values will work for other purposes.

```french
Scr # 9  AIXMODEM. 2-17-86JJ
0 \ Compute CRC
1 HEX
2 : UPDATE-CRC \ xaccum1 byte -- xaccum2
3 DATA-BYTE
4 80 1BIT 40 1BIT 20 1BIT 10 1BIT \ Apply each bit
5 08 1BIT 04 1BIT 02 1BIT 01 1BIT ;
6 DECIMAL
7 : GET-CRC \ arecord -- xrcr Test only - compute it
8 0 \ Start CRC accumulator
9 SWAP 3 - 128 OVER + SWAP DO I C0 \ UPDATE-CRC LOOP
10 0 UPDATE-CRC 0 UPDATE-CRC ;
11 : CRC-BAD? \ xrcr -- f Test for bad CRC
12 ARECORD 0 131 C0 256 \ ARECORD 0 132 C0 \ ;
13 \ Don't just fetch the CRC, because of byte-order dependency
14 1
15
```

```french
Scr # 10  AIXMODEM. 2-17-86JJ
0 \ Test for garbled data
1 : GARBLE? \ xrcr ftimeout -- fbad Test for bad record
2 \ Note: doesn't check for unexpected block #
3 \ do that at higher level.
4 TRUE \ Timed out
5 ARECORD C0 <SOH> :: OR \ No <SOH>
6 ARECORD 1+ C0 ARECORD 0 2+ C0 255 :: OR \ Bad compl
7 SWAP CRC-BAD? OR ;
8
```

```french
Scr # 11  AIXMODEM. 2-17-86JJ
0 \ Read n characters (or less if timeout)
1 : XREAD \ nexpected -- xrcr nread Return CRC, # read
2 ARECORD 0 133 0 FILL \ Avoid confusion with past data
3 BITTIMEOUT BITTIMEOUT ! \ Use block timeout on first time thru
4 0 NCHAR 0 SWAP \ Initial CRC accumulator
5 ( nexpected ) 0 DO \ Read the characters
6 XTIMEOUT XTIMEOUT CTIMEOUT ! \ Char. timeout
7 DUP 1+ IF \ Did not time out
8 DUP ARECORD 1+ C1 \ Store the character
9 I 3 < I 130 > OR IF DROP ELSE UPDATE-CRC THEN
10 \ Put only the data bytes into the CRC
11 I 1+ NCHAR \ Update count of characters
12 ELSE ( timed out ) DROP LEAVE THEN
13 LOOP 0 UPDATE-CRC 0 UPDATE-CRC NCHAR 0 ;
14
```

```french
Scr # 12  AIXMODEM. 2-17-86JJ
0 \ Check the received record
1 : XCHECK \ xrcr nread -- nflag Flag = good, 1=eof, 2=err
2 DUP 1 = ARECORD C0 <EO> AND IF \ End of file
3 DROP DROP 1 \ Return result
4 ELSE \ Regular record
5 OVER OVER 133 < ( timeout? ) GARBLE? IF \ Bad record
6 DROP DROP 2 \ Return result
7 ELSE \ Good record
8 DROP DROP 0 \ Result
9 THEN THEN ;
10 11 \ Handy words for testing
12 CREATE DATA 133 ALLOT DATA ARECORD ! \ Set up I/O area
13 : XGET -- 133 XREAD DUP \ CHKCHECK ; \ Print n flag
14 : ACK <ACK> MWRITE ; \ NAK XPURGE <NAK> MWRITE ;
15 \ To abort, use 67 MWRITE (decimal)
```

```
```
also. We want to make the words easy to use for getting in quickly and doing a job. But what happens when other options are needed? The central question is how to expand the software library in a graceful, upwardly compatible way.

One approach is to define another word, loaded with stack arguments for all the options, and then redefine MINITIALIZE in terms of it. Another approach is to define an argument-communication area for MINITIALIZE, and let it default to values which cause MINITIALIZE to behave the same unless the programmer does something with the new word. Either way allows upwardly compatible inclusion of new options, even ones not thought of in advance.

(2) It’s okay to use temporary storage areas (variables, here). The library uses a re-entrant system instead of ordinary variables, but variables are all right for now.

(3) How should routines handle errors? They must not print error messages, because they may be used when no terminal or at least no person is available. So they must signal whatever called them, so that the higher-level, application-oriented logic can decide on appropriate action.

Routines can pass an error flag to the stack. But exhaustive error reporting would make routines too cumbersome for quick and easy use. For example, MINITIALIZE could make all kinds of reports, depending on what status information was returned by the particular operating system or port.

So we took the easy approach and had MINITIALIZE throw the status and error information away. Later, it could be generalized with a new word as explained above, or by use of an error-communication area, just a place where routines dump whatever they want to report, and don’t change otherwise. Higher-level words which call the routine can, if they care, first clear the error area to null (impossible) values, then check it later to see what happened. But the user of MINITIALIZE, who doesn’t need to know about any of this, still has an easy and convenient word for ordinary uses.

Some errors fit naturally at higher levels, and should not be handled by low-level routines. For example, the word XCHECK does test whether a block number and its ones complement correspond, but it should not test whether the block number is the expected one. If it did, it would need to keep track of what record the application program was expecting and would have more complicated arguments.

Discussion — Forth Marketing

Screen 1 includes code to compile this software on the PC/Forth system mentioned above. In fact, I developed this XMODEM implementation to run on this LMI Forth, choosing it among many competent systems because it had target compilation and other support for the particular chip I was using — support which would have taken weeks to write from scratch.

This example illustrates an important aspect of the sometimes-troubled relationship between public-domain and proprietary Forth systems. Publication and software development often have different requirements.

For publication, while work in any system is welcome, it can be helpful to standardize on a publication language when possible. A large coding example will need a few words outside of the Forth standard, as we saw with KEY? vs. ?TERMINAL here; and some published papers will want to deal with internals. It would be helpful to have a common base to talk from.

It wouldn’t be fair to standardize on the system of one vendor, excluding all the others. Nor would it be wise to give one company such control over public discourse. Also, the publication language should have source code available and permit publication of parts of that code for teaching examples or in order to discuss modifications. Even if permission may be available it presents practical problems, such as the fact that you may not know specifically what to ask for until having already done the work, and then you do not know how long it would take to get a response. Using a public-domain publication language avoids these problems.

But vendors have lost sales due to the availability of free Forth systems. If they get hurt, much of the future development of Forth won’t get done. What seems to be happening is that we have passed the time when a Forth system by itself, with yet another competent editor, assembler and even target compiler, is the heart of a product. What counts more is the support of particular application-oriented tasks that certain users want to do. Potential users often complain about having to write from scratch, in Forth, tools which could have been available off the shelf if they used C. The need is there.

It is difficult to define these markets, however. When users are asked what they would want in Forth, they request vastly different and often incompatible application routines, tools or improvements. It seems that the future is in discovering application-oriented niche markets for which particular vendors can provide solid, effective support.

References


MEMBERSHIP IN THE FORTH INTEREST GROUP

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

The Forth Interest Group is a worldwide non-profit member-supported organization with over 4,000 members and 90 chapters. FIG membership includes a subscription to the bi-monthly publication, FORTH Dimensions. FIG also offers its members publication discounts, group health and life insurance, an on-line data base, a large selection of Forth literature, and many other services. Cost is $30.00 per year for USA, Canada & Mexico; all other countries may select surface ($37.00) or air ($43.00) delivery. The annual membership dues are based on the membership year, which runs from May 1 to April 30.

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

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

HOW TO USE THIS FORM

1. Each item you wish to order lists three different Price categories:
   - Column 1 - USA, Canada, Mexico
   - Column 2 - Foreign Surface Mail
   - Column 3 - Foreign Air Mail

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

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

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

FORTH DIMENSIONS BACK VOLUMES

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

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

FORML CONFERENCE PROCEEDINGS

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

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

311 - FORML PROCEEDINGS 1981 (2V) $45/48/55
   Nucleus layer, interactive layer, extensible layer, metacom-plelation, 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
BOOKS ABOUT FORTH

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

217 - F83 SOURCE . . . . . . . . $25/26/35
Dr. C. H. Ting
A complete listing of F83 including source and shadow screens.
Includes introduction on getting started.

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

219 - FORTH: A TEXT AND REFERENCE
Mahlon G. Kelly & Nicholas Spies
A text book approach to 79-Standard Forth
including section on how to teach children Forth.

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

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

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

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

224 - 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.

225 - FORTH ENCYCLOPEDIA . . . $25/26/35
Mahlon G. Kelly & Nicholas Spies
A text book approach to Forth with comprehensive references to MMS Forth and
the 79 and 83 Forth Standards.

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

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

228 - FORTH ENCYCLOPEDIA . . . $25/26/35
Mahlon G. Kelly & Nicholas Spies
A text book approach to Forth with comprehensive references to MMS Forth and
the 79 and 83 Forth Standards.

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

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

231 - FORTH ENCYCLOPEDIA . . . $25/26/35
Mahlon G. Kelly & Nicholas Spies
A text book approach to Forth with comprehensive references to MMS Forth and
the 79 and 83 Forth Standards.

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

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

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

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

236 - 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.

237 - STARTING FORTH . . . . . $22/23/32
Leo Brodie
A lively and highly readable introduction with exercises.

238 - THINKING FORTH . . . . . $16/17/20
Leo Brodie
The sequel to "Starting Forth". An intermediate text on style and form.

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

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

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 #1
Robotics. . . . . . . . . . . . . . . . . $15/16/18

402 - JOURNAL OF FORTH RESEARCH V.1 #2
Data Structures. . . . . . . . . . . . $15/16/18

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

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

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

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

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

REPRINTS

420 - BYTE REPRINTS . . . . . . . $5/6/7
Eleven Forth articles and letters to the editor that have appeared in Byte Magazine.
**DR. DOBB'S JOURNAL**
This magazine produces an annual special Forth issue which includes source-code listing for various Forth applications.

<table>
<thead>
<tr>
<th>Issue</th>
<th>Date</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>422</td>
<td>9/82</td>
<td>$5.67</td>
</tr>
<tr>
<td>423</td>
<td>9/83</td>
<td>$5.67</td>
</tr>
<tr>
<td>424</td>
<td>9/84</td>
<td>$5.67</td>
</tr>
<tr>
<td>425</td>
<td>10/85</td>
<td>$5.67</td>
</tr>
<tr>
<td>426</td>
<td>7/86</td>
<td>$5.67</td>
</tr>
</tbody>
</table>

**HISTORICAL DOCUMENTS**

<table>
<thead>
<tr>
<th>Number</th>
<th>Title</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>501</td>
<td>Kitt Peak Primer</td>
<td>$25/27/35</td>
</tr>
<tr>
<td>502</td>
<td>Fig-FORTH Installation Manual</td>
<td>$15/16/18</td>
</tr>
<tr>
<td>503</td>
<td>Using Forth</td>
<td>$20/21/23</td>
</tr>
</tbody>
</table>

**REFERENCE**

<table>
<thead>
<tr>
<th>Number</th>
<th>Title</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>305</td>
<td>Forth-83-Standard</td>
<td>$15/16/18</td>
</tr>
<tr>
<td>300</td>
<td>Forth-79-Standard</td>
<td>$15/16/18</td>
</tr>
<tr>
<td>316</td>
<td>Bibliography of Forth References</td>
<td>$15/16/18</td>
</tr>
</tbody>
</table>

**ASSEMBLY LANGUAGE SOURCE CODE LISTINGS**

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

<table>
<thead>
<tr>
<th>Number</th>
<th>Title</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>514</td>
<td>6502/Sept 80</td>
<td>$15/16/18</td>
</tr>
<tr>
<td>515</td>
<td>6800/May 79</td>
<td>$15/16/18</td>
</tr>
<tr>
<td>516</td>
<td>6809/Jun 80</td>
<td>$15/16/18</td>
</tr>
<tr>
<td>517</td>
<td>8080/Sept 79</td>
<td>$15/16/18</td>
</tr>
<tr>
<td>518</td>
<td>8086/Mar 81</td>
<td>$15/16/18</td>
</tr>
<tr>
<td>519</td>
<td>9900/Mar 81</td>
<td>$15/16/18</td>
</tr>
<tr>
<td>521</td>
<td>APPLE II/Aug 81</td>
<td>$15/16/18</td>
</tr>
<tr>
<td>523</td>
<td>IBM-PC/Mar 84</td>
<td>$15/16/18</td>
</tr>
<tr>
<td>526</td>
<td>PDP-11/Jan 80</td>
<td>$15/16/18</td>
</tr>
<tr>
<td>527</td>
<td>VAX/Oct 82</td>
<td>$15/16/18</td>
</tr>
<tr>
<td>528</td>
<td>Z80/Sept 82</td>
<td>$15/16/18</td>
</tr>
</tbody>
</table>

**MISCELLANEOUS**

<table>
<thead>
<tr>
<th>Number</th>
<th>Title</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>601</td>
<td>T-Shirt, Size</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Small, Medium, Large and</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Extra-Large. White design</td>
<td></td>
</tr>
<tr>
<td></td>
<td>on a dark blue shirt</td>
<td>$10/11/12</td>
</tr>
<tr>
<td>602</td>
<td>Poster (BYTE Cover)</td>
<td>$5/6/7</td>
</tr>
<tr>
<td>616</td>
<td>Handy Reference Card</td>
<td>FREE</td>
</tr>
<tr>
<td>683</td>
<td>Forth-83 Handy Reference</td>
<td>FREE</td>
</tr>
</tbody>
</table>

**FORTH MODEL LIBRARY**

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

**Forth-83 Compatibility IBM MSDOS**

<table>
<thead>
<tr>
<th>Vendor</th>
<th>Disk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Laxen/Perry</td>
<td>F83 LMI</td>
</tr>
<tr>
<td>MasterFORTH</td>
<td>PCIFORTH 3.0</td>
</tr>
<tr>
<td>PolyFORTH®</td>
<td>II</td>
</tr>
</tbody>
</table>

**Forth-83 Compatibility Macintosh**

MasterFORTH

**ORDERING INFORMATION**

<table>
<thead>
<tr>
<th>Number</th>
<th>Title</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>701</td>
<td>A Forth List Handler V.1</td>
<td>$40/43/45</td>
</tr>
<tr>
<td></td>
<td>by Martin J. Tracy</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Forth is extended with list</td>
<td></td>
</tr>
<tr>
<td></td>
<td>primitives to provide a</td>
<td></td>
</tr>
<tr>
<td></td>
<td>flexible high-speed</td>
<td></td>
</tr>
<tr>
<td></td>
<td>environment for artificial</td>
<td></td>
</tr>
<tr>
<td></td>
<td>intelligence. ELISA</td>
<td></td>
</tr>
<tr>
<td></td>
<td>and Winston &amp; Horn's micro-</td>
<td></td>
</tr>
<tr>
<td></td>
<td>LISP are included</td>
<td></td>
</tr>
<tr>
<td></td>
<td>as examples. Documentation</td>
<td></td>
</tr>
<tr>
<td></td>
<td>is included on the disk.</td>
<td></td>
</tr>
<tr>
<td>702</td>
<td>A Forth Spreadsheet V.2</td>
<td>$40/43/45</td>
</tr>
<tr>
<td></td>
<td>by Craig A. Lindley</td>
<td></td>
</tr>
<tr>
<td></td>
<td>This model spreadsheet first</td>
<td></td>
</tr>
<tr>
<td></td>
<td>appeared in Forth Dimensions</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Volume 7, Issue 1 and 2.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>These issues contain the</td>
<td></td>
</tr>
<tr>
<td></td>
<td>documentation for this disk.</td>
<td></td>
</tr>
<tr>
<td>703</td>
<td>Automatic Structure Charts</td>
<td>$40/43/45</td>
</tr>
<tr>
<td></td>
<td>V.3 by Kim R. Harris</td>
<td></td>
</tr>
<tr>
<td></td>
<td>These tools for the analysis</td>
<td></td>
</tr>
<tr>
<td></td>
<td>of large Forth programs were</td>
<td></td>
</tr>
<tr>
<td></td>
<td>first presented at the 1985</td>
<td></td>
</tr>
<tr>
<td></td>
<td>FORML conference. Program</td>
<td></td>
</tr>
<tr>
<td></td>
<td>documentation is contained</td>
<td></td>
</tr>
<tr>
<td></td>
<td>in the 1985 FORML Proceedings.</td>
<td></td>
</tr>
</tbody>
</table>

Please specify disk size when ordering.
# FORTH INTEREST GROUP

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Member Number</th>
<th>Company</th>
<th>Address</th>
<th>City</th>
<th>State/Prov.</th>
<th>ZIP</th>
<th>Country</th>
<th>Phone</th>
</tr>
</thead>
</table>

**OFFICE USE ONLY**

- By:  
- Date:  
- Type:  
- Shipped By:  
- Date:  
- UPS:  
- Date:  
- USPS:  
- Amt:  
- BO Date:  
- By:  
- Wt:  
- Amt:  

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

| 108 | MEMBERSHIP |  |  | SEE BELOW |

- Check enclosed (payable to: FORTH INTEREST GROUP)
- VISA
- MASTERCARD

- Card #:  
- Expiration Date:  
- Signature:  

($15.00 minimum on charge orders)

**PAYMENT MUST ACCOMPANY ALL ORDERS**

<table>
<thead>
<tr>
<th>MAIL ORDERS</th>
<th>PHONE ORDERS</th>
<th>PRICES</th>
<th>POSTAGE &amp; HANDLING</th>
<th>SHIPPING TIME</th>
<th>SALES TAX</th>
</tr>
</thead>
</table>
| Send to: FORTH INTEREST GROUP  
P.O. Box 8231  
San Jose, CA 95155 | Call 408/277-0668 to place order or for customer service. Hours: Monday-Friday, 9am-5pm PST. | 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. | Prices include shipping. A $2.00 handling fee is required with all orders. | 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). | Delivers to Alameda, Contra Costa, San Mateo, Los Angeles, Santa Cruz, and San Francisco Counties, add 6.5%. Santa Clara County, add 7%; other California counties, add 8% |

<table>
<thead>
<tr>
<th>SUBTOTAL</th>
<th>10% MEMBER DISCOUNT</th>
<th>CA. RESIDENTS SALES TAX</th>
<th>HANDLING FEE</th>
<th>MEMBERSHIP FEE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MEMBER #</td>
<td></td>
<td>$2.00</td>
<td>$30/37/43</td>
</tr>
</tbody>
</table>

**TOTAL**
On-Line Documentation

John J. Wavrik
Solana Beach, California

The best form of documentation for a Forth application is the source code. Some Forth systems provide a word LOCATE (or VIEW) so that the phrase LOCATE word will show the screen on which word has been defined. It is implemented by extending the headers of dictionary entries to include the block number on which the word was defined (and, on some systems, a file identifier).

The LOCATE word itself can be defined in a system-independent way. It is assumed that each header has been equipped with a "screen field" in addition to the usual name, link, code and parameter fields. SFA will denote the address of the screen field for the entry at hand.

Please refer to Figure One. The constant WALL marks the boundary between the words which are compiled prior to LOCATE (those which may not have the additional field) and those compiled subsequently. Several system-dependent words must be loaded before this:

```
KBT (sfa -- t = defined from keyboard)
```

The stored block number for a keyboard entry will be either 0 or -1 on most systems.

```
CFA>SFA (cfa -- sfa)
SHOW-SCREEN (sfa --)
```

This word will typically just list the screen. The user may choose to also make the screen available for editing.

```
SAVE-ENVIRONMENT
RESTORE-ENVIRONMENT
```

The LOCATE word will use a block buffer and change the contents of one or more system variables. If desired, LOCATE can be made to function transparently and restore the contents of the buffers, etc.

LOCATE is a word best supplied with the Forth system. Its ex post facto installation requires modification of the system word(s) responsible for making headers, and so a knowledge of the (system-dependent) process by which this takes place. Some examples will clarify the nature of the task. fig-FORTH mandates a word CREATE which produces headers. More recent Forth standards (Forth-79 and Forth-83) regard the manner in which headers are produced (and even the configuration of the header) to be implementation dependent. Information about the making of headers must be obtained by examination of the source code for the system, if it is available, or by decompilation. The standard defining words, CODE, CONSTANT and VARIABLE should be decomplied to identify the word responsible for making the header. In some systems, this word is an orphan (i.e., a word without a name). It will have to be decomplied using its absolute code address.

fig-FORTH

fig-FORTH is a public-domain version of Forth distributed by the Forth Interest Group starting in 1979. Its wide availability for a variety of processors made it a de facto standard.

fig-FORTH uses the word CREATE to produce new headers. (The behavior of this word is different than that in the Forth-79 and Forth-83 Standards.) The definition of CREATE begins:

```
: CREATE -find if drop ...
```

The installation of LOCATE is shown in Figure Two. When reading from the keyboard, BLK contains zero. We do not restore the environment after LOCATE.

Forth-79 Systems: MMS-FORTH v. 2.4

In this system from Miller Microcomputer Services HEAD, is the word responsible for making headers (this word was an orphan in MMS-FORTH v. 2.0). See Figure Three-b. For this system, all name fields are four bytes. The new field is eight bytes before the code field. When the system reads from the keyboard, BLK contains zero. Here we completely restore the environment after LOCATE (Figure Three-c).

In this system, data about the two buffers (numbered zero and one) is stored in an array BUFDDATA. The first two bytes of n BUFDDATA contain the number of the block currently in buffer n. The next byte indicates the order of referencing (two indicates the next buffer to be loaded). The remaining byte is the update byte (one if the block is marked, zero otherwise). SCR is a variable containing the number of the last screen listed or edited. All of this information is saved. To return blocks to the original buffers, we first identify the buffer containing the screen just listed by LOCATE. We force the next block to be loaded in this buffer. We reload the block originally in this buffer. We restore all the BUFDDATA information.

This example indicates what could be required to restore the total environment. It is more common to restore the environment only partially. If, in the midst of editing, LOCATE is used to check a definition, it is possible to ensure that the current editing block is restored to memory (but not in the original buffer and without restoring the update flag), as in Figure Three-d.

Forth-79 Systems: MVP-FORTH

The Forth available from Mountain View Press uses CREATE to make headers (Figures Four-a and Four-b). This system stores full names, as in fig-FORTH, and it uses the fig-FORTH nomenclature for accessing the fields in a header. When the system reads from the keyboard, BLK contains zero. We will make no attempt to restore the environment after LOCATE. Listing the screen will automatically make it the current screen for editing (Figure Four-c).

Forth With a File System: Kitt Peak VAX-Forth (11-Nov-82)

Here, <BUILDS is the word responsible for making headers. The definition of this word begins:

```
: <BUILDS ?ALIGN LATEST , ...
```

?ALIGN forces the dictionary pointer to the next longword boundary. Notice that Kitt Peak places the link field first, before the parameter field. This leads to the code in Figure Five-b.

Kitt Peak VAX-Forth uses the file system of the underlying operating system. Files which are being used are assigned positions in a file descriptor block table containing the file name, number of blocks, and status flags for each file. LUN is a variable holding the logical unit number (position in the table) of the currently active file. BLK contains the screen number of the current block within the current file. Both the logical unit number and the block number are saved (Figure Five-c).

BLK is -1 when the system reads from the keyboard. We save and restore only the current file. Kitt Peak's file loading word LF can be used to load a file. Its normal action is to make the file to be loaded the current file, load the file starting at block zero, close the file and restore the previous file. LOCATE requires that any file that has been used to define new words remain open. This necessitates that LF be modified not to close files and that both LF and LOAD be modified to set a file's flags so that the file cannot be closed. The redefinitions can be found in Figure Five-d. The only other change is that THEN must be replaced by ENDIF in LOCATE.
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 Forth with all currently defined words onto disk as a .COM file.

Definitions are provided to allow beginners to use Starting Forth as an introductory text.

Source code is available as an option.

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 Forth 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. $240 - $360

Up to 24kb memory: 2kb to 16kb RAM, 8k PROM contains Forth. 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
203-426-6164

Foreign orders add 95 shipping and handling. Connecticut residents add sales tax.

WORD. CR 5 SPACES HERE COUNT TYPE ;
NOT-FOUND WORD. "." is not in the dictionary " ;
TOO-EARLY WORD. "." is defined before LOCATE " ;
KEYBD WORD. "." was defined from the keyboard " ;

HERE CONSTANT WALL
LOCATE SAVE-ENVIRONMENT FIND
?DUP @= IF NOT-FOUND ELSE
DUP WALL U< IF TOO-EARLY DROP ELSE
CFA) SFA DUP KB? IF KEYBD DROP ELSE
SHOW-SCREEN THEN THEN THEN
RESTORE-ENVIRONMENT ;

Figure One

XCREATE BLK @, -FIND ;
FIND XCREATE ' CREATE !

CFA) SFA 2+ NFA 2- ;
KB? ( sfa -- true if keybd ) @ @= ;
SHOW-SCREEN ( sfa -- ) @ LIST ;
SAVE-ENVIRONMENT ;
RESTORE-ENVIRONMENT ;

Figure Two — Installation of LOCATE in fig-FORTH

VARIABLE CREATE 0, !
CONSTANT HEAD, (constant-cfa) CF, , LINKLAST !
CREATE HEAD, (variable-cfa) CF, LINKLAST ;

Figure Three-a — Decompilation

HEAD, BL WORD DUP CO .......
XHEAD, BLK @, BL ! ( patch into existing word )
FIND XHEAD, ' HEAD, !

Figure Three-b — Partial definition of MMS's HEAD.

CFA) SFA 2- !
KB? ( sfa -- true if keybd ) @ @= !
SHOW-SCREEN ( sfa -- ) @ LIST ;
1 DARRAY SVDATA VARIABLE SVSCR
SAVE-ENVIRONMENT 2 0 DO I BUFFDATA 20 I SVDATA 2!
LOOP SCR @ SVSCR ! ;
RESTORE-ENVIRONMENT SCR @ SVSCR @ SCR !
BUFFDATA @ = @ ELSE 1 @ THEN
BUFFDATA 2+ 1 SWAP C!
BUFFDATA 2+ 2 SWAP C!
SVDATA @ DUP @ NOT IF BLOCK THEN DROP
2 0 DO I SVDATA 20 0 I BUFFDATA 2! LOOP 1

Figure Three-c
Figure Three-d

| SAVE-ENVIRONMENT SCR @ SVSCR ! |
| RESTORE-ENVIRONMENT SVSCR @ DUP SCR ! BLOCK DROP |

Figure Four-a — Decomposition

| CREATE BL WORD DUP DUP .... |
| XCREATE BLK @ , BL ! (patch into existing word) |
| FIND XCREATE \ CREATE |

Figure Four-b — Partial definition of MVP's CREATE

| CREATE 2 ALLOT ! |
| CONSTANT CREATE , , CODE ... |
| CODE CREATE SMUDGE HERE DUP 2- ! ENTERCODE |

Figure Four-c

| CREATE BL WORD DUP DUP .... |
| XCREATE BLK @ , BL ! (patch into existing word) |
| FIND XCREATE \ CREATE |

Figure Five-a — Decomposition

| X(BUILDS ?ALIGN LUN @ B, BLK @ B, B @ B) FIND X(BUILDS \ BUILDS ! |

Figure Five-b — Making headers in Kitt Peak's file system

| CFA>SFA IA + PFA>NFA 8 - |
| KB? (sfa -- true if keybd) | 0= |
| SHOW-SCREEN (sfa -- ) @ LIST |
| SAVE-ENVIRONMENT |
| RESTORE-ENVIRONMENT |

Figure Five-c

| "LF LUN @ ) R 1 FTU.BIT OR DOFOPEN ?FILE LUN ! CR @ 0 .LINE 0 LOAD LUN @ FFLAGS DUP @ DUP FTU.BIT AND IF FTU.BIT COM AND ENDIF 3 COM AND 1 OR SWAP \ (R LUN | |
| LF (LF filename) GBN "LF |
| LOAD (# -- ) LUN @ FFLAGS DUP @ 1 OR SWAP ! LOAD |

Figure Five-d

---

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

**THREE TOUGH QUESTIONS WITH ONE EASY ANSWER:**

1. **WHEN IS A COMPUTER LANGUAGE NOT A LANGUAGE?** MMSFORTH includes DOS, Assembler and high level commands and extraordinary utilities, extends to become any other language (or application), is an interpreter and a compiler, and is remarkably fast and compact!

2. **WHICH SOFTWARE RUNS THE SAME DISKS IN IBM PC AND TRS-80 MODEL 4?** MMSFORTH disks run on those and Compaq, and TRS-80 Model 3, and Tandy 1200, and TRS-80 Model 1, and AT&T 6300, etc., with your choice of formats up to 200K single-sided or 400K double-sided!

3. **WHO OFFERS SOURCE CODE WITH ITS LANGUAGE, UTILITIES, DATABASE, WORD PROCESSOR AND COMMUNICATIONS SOFTWARE?** Nearly all MMSFORTH software includes source code.

---

**MMSFORTH**

All the software your computer may ever need.

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

- **Personal License (required):** MMSFORTH Disk Disk 80 System Disk (TRS-80 Model 1 requires 1.44MB, £59.95)
- **Personal license (additional modules):** FORTHCOMM communications module £ 49.95
**Utilities**: £49.95
**Games**: £39.95
**EXPERT-2 expert system**: £69.95
**DATANUT**: £59.95
**DATANUT PLUS**: (PC only, 128K req) £99.95
**FORTHWRITE word processor**: £99.99
**Corporate Site License Extensions** from $1,000
**Bulk Distribution**: from $500/50 units.

- **Some recommended FORTH books:**
  - **STARTING FORTH (programming)** £9.95
  - **THINKING FORTH (technique)** £15.95
  - **BEGINNING FORTH (in MMSFORTH)** £18.95

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

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

---
Eighth Annual

Forth National Convention

November 21–22, 1986

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

Conference Program

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

Demonstrations • New Products • Tutorials • Chapter Activities

Forth Exhibits

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

<table>
<thead>
<tr>
<th>Event</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Convention preregistration</td>
<td>$15</td>
</tr>
<tr>
<td>Registration at the door</td>
<td>$20</td>
</tr>
<tr>
<td>Banquet Saturday 7 p.m.</td>
<td>$35</td>
</tr>
</tbody>
</table>

Conference and Exhibit Hours
Friday Nov. 21st 12 noon—6 p.m.
Saturday Nov. 22nd 9 a.m.—5 p.m.

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

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

TOTAL CHECK TO FIG $__________

Name ____________________________
Address __________________________
Company __________________________
City _____________________________ State ________ Zip _______________________
Phone (_______) ____________________

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

Gary Smith
Little Rock, Arkansas

Daily more individuals are joining the Forth community, and they bring a need for information. Fortunately, most have modems and supporting telecommunications software. For at the same time, more Forth information than ever before is available on private and vendor-supported electronic bulletin boards, and also on some electronic information services like CompuServe and BIX.

The following information is intended to bridge the gap that presently exists between the user in need and the resource waiting to serve. I am satisfied that I have not listed all possible resources and in the dynamic world being discussed here, there are certain to be changes. If readers help to supply additional information, or pose specific questions about electronic sources of Forth-related information, this feature can be updated as required. Just write to Forth Dimensions in a letter separate from any other FIG business; and be sure to include your electronic address, if you have one!

My very sincere thanks to Jerry Shifrin of the East Coast Forth Board for his help in compiling and verifying much of the information published here. His board is included in the list, and is truly one of the best of the listed resources.

All that follows was accurate (or reasonably so) at the time this article was written. The author, Forth Dimensions and the Forth Interest Group assume no responsibility for any omissions, deletions or errors.

BULLETIN BOARDS

<table>
<thead>
<tr>
<th>Name/Provider</th>
<th>Phone No.</th>
<th>Hours</th>
<th>Baud/Protocol</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>The FIG Tree</td>
<td>415/538-3580</td>
<td>24 hours</td>
<td>300</td>
<td>CommuniTree BBS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>7 days/wk.</td>
<td>7 bits, even parity</td>
<td>Messages only. Lots of info. Type 2 CRs to begin.</td>
</tr>
<tr>
<td>Thousand Oaks</td>
<td>805/492-5472</td>
<td>24 hours</td>
<td>parity</td>
<td>Files supported.</td>
</tr>
<tr>
<td>Technical BBS</td>
<td></td>
<td>7 days/wk.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>East Coast Forth</td>
<td>703/442-8695</td>
<td>24 hours</td>
<td>300,1200,2400</td>
<td>Registration req., but free. Very full service.</td>
</tr>
<tr>
<td>Board</td>
<td></td>
<td>7 days/wk.</td>
<td>8, 1, none</td>
<td>Much from others ported here.</td>
</tr>
<tr>
<td>The Forth Source</td>
<td>215/630-9149</td>
<td>2pm-8am EST</td>
<td>300</td>
<td>Jon Day (sysop) Forth conf., but no up/down load of files.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>24 hrs. on weekend</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LMI BBS</td>
<td>213/306-3530</td>
<td>6pm-9am PST</td>
<td>300,1200,2400</td>
<td>Wes Meier BBS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>24 hrs. on weekend</td>
<td>8, 1, none</td>
<td>Support to users by Laboratory Microsystems. Others NOT restricted from use.</td>
</tr>
<tr>
<td>Correct Software</td>
<td></td>
<td></td>
<td>300,1200</td>
<td>Currently down. Non-users of Correct products send $15.00 for registration to: Rt. 1, Box 140 Black Hawk, SD 57718</td>
</tr>
<tr>
<td>The WELL</td>
<td>415/332-6106</td>
<td>24 hrs.</td>
<td>300,1200,2400</td>
<td>UNIX system conference **NOTE: Use lower case! Gateway to USENET via GO TELE then choose #13. Have VISA/MC ready and type newuser (lower) Mike Ham is Forth Guru here.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>7 days/wk.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Computer Language</td>
<td>415/957-9370</td>
<td>24 hrs.</td>
<td>300,1200</td>
<td>FIDO BBS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>7 days/wk.</td>
<td></td>
<td>Extensive message base. Many files.</td>
</tr>
</tbody>
</table>
polyFORTH GETS YOUR PROGRAM FROM CONCEPT TO REALITY 4 TO 10 TIMES FASTER

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

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

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

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

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

For more information and a free brochure, contact FORTH, Inc. today:

FORTH, Inc., 111 N. Sepulveda Blvd., Manhattan Beach, CA 90266. Phone (213) 372-8493.

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

HS/FORTH

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

HS/FORTH, complete system: $395.

Information Systems

CompuServe (Call 800-848-8990 for information.) CompuServe features three conferences with extensive FORTH coverage. Summaries are as follows:

"CLM Conference" Similar to The WELL and Computer Language's own BBS, and operated by Computer Language magazine. DL-7 is the FORTH library.

Go CLM at the ! prompt.

"Computer Solutions, Inc." Emphasis on CSI products MacForth and MultiForth, but also lots of goodies and discussion for the general FORTH crowd. DL-6 is the FORTH library.

Go PSG-4 or Go FORTH at the ! prompt.

"Dr. Dobb's Journal" This is the newest entry and is not yet as active as the previous two. With Ray Duncan and Michael Ham as syops, that will change! DL-6 is the FORTH library.

Go DDJFOR at the ! prompt.

The Special Interest Groups for the Model 100 and Commodore also have some excellent machine-specific FORTH items. Commodore has Blazin' Forth (a public-domain FORTH-83 implementation) and M-100 has a public-domain FORTH that supports Model 100s using the Chipmunk drive.

BIX (Call 800-227-2983 for information.) BYTE Information Exchange is BYTE magazine's conference system. One of the hundred or so sections is the FORTH conference with Phil Wasson as syop. This is a good source for hot-breaking news. There is a separate conference area for files.

JOIN FORTH at the ! prompt.

There is also considerable FORTH discussion in appropriate conferences such as 4TH.GEN.LANGS but it must be sought out.

References

1. CompuServe

5000 Arlington Centre Blvd.

Columbus, OH 43220

(or inquire at your local dealer)

2. BYTE Information Exchange

70 Main Street

Peterborough, NH 03458
Forth Source Formatter

John Konopka
Miakko Shi, Japan

It is considered good form in any computer language to keep one's source code neatly formatted. There are various recommendations about when to indent, when to start a new line, etc. The problem with this is that it takes too much effort. It's like asking a teenager to keep his room clean: nice in theory, but difficult to implement. The problem is accentuated for Forth programmers editing existing source who are asking a teenager to keep his room clean:

The problem is accentuated for Forth programmers editing existing source who are asking a teenager to keep his room clean: one screen. While we don't yet have robots which will clean our rooms for us, we do have a robot which can make source code legible: the computer.

The program presented here will take the most compact, compacting source code as input and will turn out a cleanly formatted listing with new lines for every colon definition and proper indenting for structured constructs. A fill algorithm is used to prevent words from running off the right edge of the page. Parameters are available to adjust the width and length of the listing to any printer or terminal. Line feeds are counted so that form feeds can be generated at appropriate intervals.

In operation, the program works similarly to a word processor. Instead of adding “dot commands” to the source to cause various actions, special Forth words are interpreted as commands to automatically cause formatting of the text. The code for the main loop is in screen #7. It parses one word of text, then checks if this is a special word. If it is, then some formatting occurs; if not, the word is simply typed. To avoid storing strings of text and doing string comparisons, FIND is used to get one word of source then convert this to an address. This address is compared against the stored addresses by the word ?? when checking for special words. Rather than assume anything about what FIND does to a string or where it leaves it, the value of >IN is saved before invoking FIND and then is restored in TYPE-IT before parsing the word again with WORD.

Screen #2 contains the parameters controlling the dimensions of the listing. The constants LINE-SIZE and MAX-LINES control the width and page length respectively. By changing these, you can match the listing to your printer or screen size. The constant OFFSET determines how far the margin is moved whenever an indent occurs. The value of three seems good, but you might want to use two or even one if you work with a very narrow page. Variable COLUMN is the count-

er containing the current character distance from the left edge of the page. Variable TAB is the left margin measured in spaces from the left edge of the page. Variable LINES counts the number of lines typed since the last form feed.

The arrays of stored addresses are held in screen #3. An array is created for each action desired. For example, IF and DO both require an indent and a new line, so they are placed in the array called INS. If you define new branching words you can add them to one of these arrays. The first integer in the array says how many addresses are in the array.

INDENT, OUTDENT and NEW-LINE (defined in screen #4) move the left margin in the fashion suggested by their names. Two other special formatting words are defined in screen #6. These are TYPE-TO- and TYPE-TO-END-OF-LINE. The first emits words until it encounters the character it is given on the stack. It is used to list out a series of words such as those between quotes or parentheses. The second emits text until >IN is an integer multiple of 64. Its purpose is to list comments set off by a backslash. This second word is an exception in that it types directly out of the text buffer and it does not check string length before typing. Both of these list text without moving the margin, even though the typed string contains words such as DO, LOOP, IF, THEN, etc.

Two other words, CR? and TYPE-IT (screens #4 and #5 respectively), are used to send all text to the output device (with the one exception noted above). CR? performs a carriage return and line feed only when needed, avoiding blank lines in the output. It also counts the number of line feeds emitted and emits a form feed when the number of lines exceeds the value in constant MAX-LINES. TYPE-IT restores >IN from temporary storage in IX and then parses a string with WORD. It compares the length of each word to be typed with the remaining space on the line, then invokes CR? if there is insufficient room to type the word. It's possible to type off the right edge of the page if there is still insufficient room to type the word even after invoking CR?. This can happen with deeply nested IF ELSE THEN clauses and long names.

<table>
<thead>
<tr>
<th>Screen #2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 \ F=L Variables and Constants</td>
</tr>
<tr>
<td>2 VARIABLE COLUMN \ Current column pointer.</td>
</tr>
<tr>
<td>3 VARIABLE TAB \ Position of left margin.</td>
</tr>
<tr>
<td>4 VARIABLE LINES \ Count of lines typed since last FF.</td>
</tr>
<tr>
<td>5 VARIABLE FIND \ Flag indicating currently parsed word was</td>
</tr>
<tr>
<td>6 VARIABLE IX \ Temporary storage of value of &gt;IN.</td>
</tr>
<tr>
<td>7 VARIABLE IX \ Temporary storage of value of &gt;IN.</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>9 79 CONSTANT LINE-SIZE \ Column width of output device.</td>
</tr>
<tr>
<td>10 65 CONSTANT MAX-LINES \ Lines per page of output device.</td>
</tr>
<tr>
<td>11 3 CONSTANT OFFSET \ Size of indentation in columns.</td>
</tr>
<tr>
<td>12 34 CONSTANT A&quot; \ ASCII value of double quote.</td>
</tr>
<tr>
<td>13 41 CONSTANT PAREN \ ASCII value of right parenthesis.</td>
</tr>
<tr>
<td>14</td>
</tr>
<tr>
<td>15 FIND \ CONSTANT BSLASH</td>
</tr>
<tr>
<td>16 FIND ( CONSTANT PAREN</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Screen #3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 \ F=L Arrays of pointers to formatting words.</td>
</tr>
<tr>
<td>2 CREATE QUOTES 2 , FIND , , FIND :</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4 CREATE STARTS 2 , FIND : , FIND CODE</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>6 CREATE ENDS 3 , FIND ; , FIND NEXT , , FIND END-CODE ,</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>8 CREATE OUTS 5 , FIND UNTIL , FIND LOOP , FIND +LOOP</td>
</tr>
<tr>
<td>9</td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td>11 CREATE INS 3 , FIND IF , FIND DO , FIND THEN</td>
</tr>
<tr>
<td>12</td>
</tr>
<tr>
<td>13 CREATE IN+OUTS 4 , FIND ELSE , FIND WHILE</td>
</tr>
<tr>
<td>14</td>
</tr>
<tr>
<td>15</td>
</tr>
<tr>
<td>16</td>
</tr>
</tbody>
</table>
All the parts needed to make the
SMALLEST PROGRAMMABLE
FORTH SYSTEM:

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

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

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

Availability: F68HC11 Production units
with Max-FORTH™ in internal ROM avail-
able 40Q86. Volume quantity available
1Q/87. X68HC11 emulator with Max-
FORTH™ in external ROM available
now. NMIX-0022 68HC11 Development
System boards available now: $290.00.

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

### Screen 4

```
1 \ F+L CR? INDENT OUTDENT NEW-LINE
2 \ --- Perform a carriage return (maybe) and form feed (maybe).
3 \ Don't do CR on new lines. This avoids blank lines.
4 : CR? COLUMN @ TAB @ > \
   \ Legitimate CR ?
5 \ IF CR @ LINES @ MAX-LINES = \ Do CR, need a FF ?
6 \ IF SFF 0 @ LINES ! \ Do FF
7 \ ELSE 1 @ LINES +1 ! \ Count lines
8 \ THEN 0 COLUMN ! THEN ; \ Reset column pointer.
9 \ n --- Indent by constant OFFSET times n.
10 : INDENT CR? OFFSET * TAB +1 ;
11 12 \ n --- Move margin left OFFSET times n. Don't move off page.
13 : OUTDENT CR? TAB @ SWAP OFFSET * - DUP 0 ;
14 \ IF BELL THEN 0 MAX TAB ! ;
15 \ --- Emit by constant OFFSET times n.
16 : NEW-LINE 0 TAB 1 CR? ;

Screen 5

```

```
1 \ F+L DONE? PARSE TYPE-IT SFF
2 \ --- f1 Check for end of screen.
3 \ DONE? >IN @ 1023 ;>

5 \ --- Place next word at HERE ready for typing.
6 \ PARSE IX @ >IN : BL WORD DROP ;
7 
8 \ --- Type one word, first do CR if no room on this line.
9 \ TYPE> COLUMN @ TAB @ IF TAB @ COLUMN @ SPACES TAB @ COLUMN ! THEN HERE COUNT + DUP COLUMN @ + LINE-SIZE >
10 \ IF CR@ TAB @ SPACES TAB @ COLUMN ! THEN DUP
11 \ COLUMN +1 TYPE -1 FND ! ;
12 \ ---
13 \ TYPE-IT PARSE <TYPE> ;
15 \ --- Emit a form feed.
16 \ SFF 12 EMIT ;

Screen 6

```

```
1 \ F+L ?? F+L-INIT TYPE-TO- TYPE-TO-END-OF-LINE
2 \ x y -- x f1 Flag true if x found in array at address y.
3 \ ?? 0 SWAP DUP SWAP @ 0 DO DO @ 4 PICK =
4 \ IF SWAP 1+ SWAP LEAVE THEN LOOP DROP ;
5 \ n --- m p List block n. Save BLK and >IN, restore later.
7 \ F+L-INIT BLK @ >IN @ ROT BLK ! 0 >IN ! 1 @ LINES !
8 \ 0 TAB 1 COLUMN ! " Screen #" BLK @ . CR ;
9 \ n --- Type text till next occurrence of n.
11 \ TYPE-TO- CR? TYPE-IT WORD C@ +1 HERE C! BL HERE DUP C@
12 \ 1+ C! <TYPE> CR? ;
13 \ --- Type comment line, round up >IN.
14 \ TYPE-TO-END-OF-LINE CR? TYPE-IT BLK @ BLOCK >IN @ >IN @ 63
15 \ COM AND 64 + DUP R >IN @ - TRAILING TYPE CR? R >IN ! ;
16
```
Screen #7
1 $F+L$ END
2 \n --- Formatted listing of screen #n.
3 $F+L$ $F+L$-INIT
4 BEGIN \ FIND \ STARTS ?? IF NEW-LINE TYPE-IT \ 0 \ IX ! THEN
5 QUOTES ?? IF A" TYPE-TO- THEN
6 DUP PAREN = IF RPAREN TYPE-TO- THEN
7 DUP BSLASH = IF TYPE-TO-END-OF-LINE THEN
8 STARTS ?? IF NEW-LINE TYPE-IT >IN 0 IX ! THEN
9 TYPE-IT 2 INDENT THEN
10 INS ?? IF 1 INDENT TYPE-IT 1 INDENT THEN
11 OUTS ?? IF 1 OUTDENT TYPE-IT 1 OUTDENT THEN
12 ENDS ?? IF 2 OUTDENT TYPE-IT CR? THEN
13 IN+OUTS ?? IF 1 OUTDENT TYPE-IT CR 1 ELSE 0 THEN
14 DROP DONE? IF NEW-LINE TYPE-IT CR SFF 1 ELSE THEN
15 FND 0 NOT IF TYPE-IT THEN
16 UNTIL >IN ! BLK ! ;

Source code before formatting.

Screen #8
1 $F+L$ END
2 \n --- Formatted listing of screen #n.
3 $F+L$ $F+L$-INIT
4 BEGIN \ FIND \ STARTS ?? IF NEW-LINE TYPE-IT \ 0 \ IX ! THEN
5 QUOTES ?? IF A" TYPE-TO- THEN
6 DUP PAREN = IF RPAREN TYPE-TO- THEN
7 DUP BSLASH = IF TYPE-TO-END-OF-LINE THEN
8 STARTS ?? IF NEW-LINE TYPE-IT >IN 0 IX ! THEN
9 TYPE-IT 2 INDENT THEN
10 INS ?? IF 1 INDENT TYPE-IT 1 INDENT THEN
11 OUTS ?? IF 1 OUTDENT TYPE-IT 1 OUTDENT THEN
12 ENDS ?? IF 2 OUTDENT TYPE-IT CR? THEN
13 IN+OUTS ?? IF 1 OUTDENT TYPE-IT CR 1 ELSE 0 THEN
14 DROP DONE? IF NEW-LINE TYPE-IT CR SFF 1 ELSE THEN
15 FND 0 NOT IF TYPE-IT THEN
16 UNTIL >IN ! BLK ! ;

Source code after formatting.

MasterFORTH

FORTH-83 STANDARD

- 6809 Systems available for $150
- 6809/6809 ........... $150
- 680x0 Systems available for $125
- MACINTOSH ................ $150
- FORTH/20 for 68020 Single Board Computer

Disk based development system under OS9/68K ........ $290
EpROM set for complete stand-alone SBC .......... $390
- Forth Model Library - List handler, spreadsheet, Automatic structure charts ... each $40
- Target compilers : 6809,6801, 6303, 680x0, 8088, 280, 6502

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

HARDWARE for the HARDCORE

68020 SBC, 5 1/4" floppy size board with 2MB RAM, 4 x 64K
EpROM sockets, 4 RS232 ports,
Centronics parallel port, timer, battery backed date/time,
interface to 2 5 1/4" floppy and a SMI interface to 2
winchester disks .......... $2750
68881 f/p option .......... $500
OS9 multitask&user OS .. $350

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

Mike Elola
San Jose, California

Decomposing functions is a critical part of Forth programming. Dual-CFA definitions provide the benefits of decomposed functions in situations where decomposition would not be possible normally. Besides requiring two CFAs, this kind of functional decomposition is implemented in reverse of the normal dictionary order. (Normally, Forth requires a bottom-up ordering of definitions — child definitions are compiled ahead of the parent definitions that reference them.)

By first exploring more conventional Forth programming techniques, various issues will emerge that showcase the advantages of dual-CFA decomposition.

Illustrative Example

Many dual-CFA definitions could be replaced with similar definitions that use vectored execution. Unfortunately, the use of execution vectors can negate certain advantages of normal Forth decomposition.

Suppose that you have revectored TYPE to route its output to the printer. A pause is desired to allow users to insert another sheet of paper, so you define the following word:

```
: FEEDPRINTER ( -- )
CR . " Insert new sheet. Press any key to continue." KEY DROP ;
```

When run, the message is typed at the printer rather than displayed at the user terminal. TYPE is called by (\) which is compiled by . within the definition of FEEDPRINTER.

The source of this problem may be the decomposition itself. The revectored version of TYPE shows distinct I/O redirection behavior. In normal development, the enlargement of TYPE's function would be implemented in a new and separate definition incorporating TYPE. The revectored TYPE cooperate with its parent definition. Thus, higher-level words such as FEEDPRINTER could engage the correct version of TYPE upon compilation. There would be no possible chance for erroneous, surprise behavior.

Including the function of output redirection within TYPE integrates more functionality into the word than was originally intended. This prevents a more complete factoring of these related but distinct functions. As such, TYPE is a poorly decomposed function. Because of the use of a vector, less consideration than usual is given to the decomposition involved. (This is something to stay on guard for when execution vectors are used in an application.)

Compensatory code can be formatted to solve the problems associated with this example. For instance, FEEDPRINTER can buffer the contents of the vector variable, reset it to a value that causes output to the screen, write the message, then reinitialize the original vector value.

However, such remedies add complexity to the task of programming and increase the chance of programming error. When the behavior of a word must be variable, extra code is required each time it is used to ensure that the correct behavior will be selected when it is run. Ultimately, a heavy price is paid when compiling words with variable (loosely-coupled) actions.

This example is not presented to suggest that vectored execution is never advantageous. The run-time flexibility of vectored execution is justification enough for its use. But when the desired behavior can be selected at compile time, a better programming solution is possible.

Preferred Forth Style

Forth is at its best when variable-behavior definitions are avoided. Functions have to be well decomposed when fixed-behavior words are used exclusively. Together, such words impart hardiness to an application by eliminating dependence on the current environment.

As an extensible programming language, Forth can exhibit a wide range of functionality: with each new word added, its functionality is extended. In this way, Forth can provide a wide range of functions to better deal with a wide range of programming problems. If the extensions are fixed-behavior words, Forth offers superior ease of programming as well.

A Forth programming philosophy aimed at memory compactness, brevity of expression and ease of use is realized when words in higher memory consistently integrate more functionality than those located in lower memory. Lower-level words should exhibit decreasing functional scope. All words should perform single, fixed behaviors.

To achieve this, each new higher-level definition should consolidate more functionality. This way, each word in a common execution path represents an increment of progress toward the overall application function. Progress toward the ultimate goal that is not incremental may create problems and usually indicates incomplete decomposition.

Programs developed in this way exhibit the following characteristics:

1. Compactness of compiled code
2. Fewer conditional phrases
3. Single behaviors per word

The following three sections explore the effectiveness of other, more conventional techniques that in some ways comply or do not comply with this programming philosophy.

Deferred Definitions

Normally, the behavior of a deferred word is not intended to be variable. Unanticipated behavior is only possible when deferred words are incorrectly initialized. Once initialized, deferred words should provide the same benefits as fixed-behavior words.

---

**Figure One**

<table>
<thead>
<tr>
<th>EMIT</th>
<th>CR</th>
<th>CR'S</th>
</tr>
</thead>
<tbody>
<tr>
<td>Child</td>
<td>Parent</td>
<td>Child</td>
</tr>
</tbody>
</table>

Dictionary sequence for standard colon definitions.

<table>
<thead>
<tr>
<th>CR</th>
<th>ITERATOR:</th>
<th>CR'S</th>
</tr>
</thead>
<tbody>
<tr>
<td>Child</td>
<td>Parent</td>
<td>Child</td>
</tr>
</tbody>
</table>

Dictionary sequence that includes dual-CFA definitions.
To ascertain that one word represents uniformly increasing functionality relative to others, it is necessary to examine its execution path. (It is not relevant simply to compare a word’s physical location with that of its parents.) Consequently, deferred words and dual-CFA words can both be used effectively to create a sequence of words where each subsequent one consolidates slightly more functionality.

** Vectored Definitions **

Frequently, vectored definitions are employed to create words with variable behaviors. As such, their use should be avoided if at all possible. The first example given (FEED-PRINTER) helped demonstrate the problems that arise when vectored definitions disguise poor decomposition. Still other problems are directly associated with the use of vectored execution (as well as with other variable-behavior definitions).

While vectored definitions can execute rapidly, they also create additional work. An effort must be made to maintain the correct “current” value of a vector, particularly when it is dynamically changing. This usually takes the form of “housekeeping” initializations and finalizations with each use of the vectored word. The extra code can obscure the underlying algorithm.

Another problem with vectored definitions is increased vulnerability to error—particularly when one revectoring word calls another. Unless a stack is used to help maintain the correct value of the execution vector, the higher-level word has no guarantee that its initializations of vectors remain in effect after other words are called.

Still, memory compactness is sometimes served best through vectored execution. Redirection of input or output is a good example of this. Revectoring EMIT is more memory efficient than generating custom versions of EMIT, CR, SPACE and TYPE for every possible output device.

While vectoring enables run-time flexibility and seems to be the shortest path to a coded solution, the support required to deal with the added complexity must be considered. A variable-behavior EMIT may create the need for fixed-behavior versions of output words just to make programming easier. For instance, if there is a need for FEED-PRINTER, then it alone is justification for SCREEN-TYPE, a fixed-behavior word. Ultimately, a mix of fixed- and variable-behavior words may prove most effective in situations like this.

**Other Variable-Behavior Definitions**

The behavior-binding function of the Forth compiler is rendered inadequate whenever variable word behaviors are allowed. This is true however the variable behaviors are ultimately selected. Vectored definitions are one form of variable-behavior words. Other kinds of words are also used to implement variable behaviors.

A way to select one of several actions is through the use of a conditional phrase. Consider the following definition for TYPE:

: TYPE ( add count -- )
  OUT-DEV @ IF ( printer code . . . )
  ELSE ( screen code . . . ) THEN ;

The behavior of this word is under the control of the variable OUT-DEV. Now, before and after each use of TYPE the programmer must decide how OUT-DEV should be initialized and finalized. Note that the same burdens of environmental maintenance befall the use of this definition as would a vectored definition. Only now, OUT-DEV is the environmental variable requiring extra care, instead of a vector-containing variable.

The following is an attempt to improve on the previous version:

: TYPE ( add count dev -- )
  IF ( printer code . . . )
  ELSE ( screen code . . . ) THEN ;

Now the problems with maintaining the current environment disappear because the behavior-selection mechanism is one of TYPE’s input parameters. The drawbacks that remain are the extra execution overhead and memory overhead for the conditional phrase, as well as the extra code necessary to generate the additional stack item consumed with every call to TYPE. All of these drawbacks could be avoided if a fixed-behavior, screen-oriented TYPE were available (assuming that it could be selected at compile time).

(When run-time flexibility is needed, there is often no choice but to use a variable-behavior word. Such is the case when user-selectable actions are desired. The Forth interpreter manages to accomplish this admirably, since it is very compact and employs very few conditionals.)
Dual-CFA Definitions

Deferred definitions, vectored definitions and other variable-behavior definitions have been considered so far. However, dual-CFA definitions have yet to be separately presented. For the purpose of discussion, suppose that a definition of ITERATOR already exists, and that it is a defining word that can be used to produce dual-CFA definitions. The first CFA compiled points at an iteration function that can be the parent to many similar words, such as CR'S and SPACES:

ITERATOR: CR'S CR ;
ITERATOR: SPACES SPACE ;

An equivalent colon definition for CR'S is:

CR'S 0 DO CR LOOP ;

These definitions will help illustrate the merits of dual-CFA definitions in terms of the programming family already presented.

The dual-CFA definitions are obviously much shorter. This is possible because the common code for CR'S and SPACES has been factored into the parent defining word ITERATOR:

ITERATOR: create docol , compile-def does> ;
SWAP 0 ( cfaz #times 0 —— )
DO DUP EXECUTE LOOP DROP ;

Other support words must be added to allow recursion within the parent portion of dual-CFA definitions. Still other support words, or altogether different implementations, are needed to support multi-level, dual-CFA decomposition.

Other dual-CFA defining words have been found useful include:

TREE-TRAVERSER: (with TREE-BALANCE, TREE-SCAN and TREE-PRINT), EXEC-COUNT, CALL-TRACE, and RELOCATABLE.
Conclusions

The Forth language suggests a philosophy of decomposition that rewards well those who find ways to conform with it. Compliance involves compiling fixed-behavior words. Each new word is defined to consolidate slightly more functionality than the previous one in the same execution path.

Dual-CFA definitions can uphold this philosophy when other kinds of definitions do not. Particularly in those situations where a deviation in dictionary sequence is required, this technique reaps all the benefits normally reserved for Forth words implemented in the standard child-before-parent sequence.

The Forth language generates enthusiasm among programmers by offering the possibility of very streamlined code. Considering the prospects for dual-CFA decomposition, this enthusiasm can be expected to increase.

Index to Advertisers

Bryte – 10
Computer Cowboys – 12
Dash, Find & Associates – 13
Forth, Inc. – 26
Forth Interest Group – 7, 17–20, 24
Harvard Softworks – 26
Laboratory Microsystems – 14
MCA – 22
MicroMotion – 31
Miller Microcomputer Services – 23
Mountain View Press – 6
New Micros – 28
Next Generation Systems – 33
Palo Alto Shipping Company – 4
Software Composers – 2
SOTA – 32
Talbot Microsystems – 29
Tools Group – 8
Ubz Software – 11

Mr. Apra's suggestion for experiments with conditional structures as a source of programming exercises for students.

Forth offers a unique choice: one can choose programming languages that incorporate a large number of "features" (which can't be changed and which may be hard to use or to remember) or one can choose a programming language which is conceptually simple and allows the user to incorporate a wide variety of features on his own. Understanding is a powerful "handle" which I would not want to trade for "features." I echo Mr. Apra's "Keep it simple," but add the words "... in concept."

Sincerely yours,

John J. Wavrik
U.C. San Diego, California

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

STANDARD FEATURES

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

- **Oklahoma City, OK**
  73112

- **OCTON**
  Dayton, OH
  Call Gary M. Granger
  513/849-1463

- **Pennsylvania**
  Philadelphia Chapter
  Monthly, 4th Sat., 10 a.m.
  Drexel University, Stratton Hall
  Call Melanie Hoag or Simon Edwards
  215/895-1483

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

- **Texas**
  Dallas/Ft. Worth
  Metroplex Chapter
  Monthly, 1st Sat., 7 p.m.
  Call Tom Almy
  503/692-28

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

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

- **Vermont**
  Vermont Chapter
  Monthly, 1st Sat., 7:30 p.m.
  Vergennes Union High School
  Rm. 210, Monkton Rd.
  Vergennes, VT
  Call Don Van Syckel
  802/388-6698

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

- **Virginia**
  Potomac Chapter
  Monthly, 1st Sat., 7 p.m.
  Lee Center
  415 Business School
  Univ. of Richmond
  Call Donald A. Full
  704/898-4099

### Wisconsin
- **Wisconsin**
  Lake Superior FIG Chapter
  Monthly, 2nd Sat., 7:30 p.m.
  University of Wisconsin Superior
  Call Allen Anway
  715/394-8560

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

### England
- **England**
  Forth Interest Group — U.K.
  Monthly, 1st Sat., 1 p.m.
  Contact D. Neale
  800 Oak Ridge
  Madison, WI 53705

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

### Holland
- **Holland**
  Holland Chapter
  Contact: Adriaan van Roosmalen
  Heusden Houtestraat 134
  4817 We Breda
  31 76 713104

### Canada
- **Canada**
  Alberta Chapter
  Call Tony Van Muyden
  403/962-2203

### Ireland
- **Ireland**
  Irish Chapter
  Contact Hugh Doggs
  Newton School
  Waterford
  051/75757 or 051/74124

### Italy
- **Italy**
  FIG Italia
  Contact Marco Tausel
  Via Gerolamo Forini 48
  20161 Milano
  02/645-868

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

### Norway
- **Norway**
  Bergen Chapter
  Kjell Birger Feiraas
  804/962-2203

### Republic of China
- **Republic of China**
  R.O.C.
  Contact Ching-Tang Tzeng
  P.O. Box 28
  Lung-Tan, Taiwan 325

### Sweden
- **Sweden**
  Swedish Chapter
  hans Lindstrom
  Gothenburg
  +46-31-166794

### Switzerland
- **Switzerland**
  Swiss Chapter
  Contact Max Hugelshofer
  416/377-0708

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

### FIGGRAPH
- **FIGGRAPH**
  Call Howard Pearlmutter
  408/425-8700

---

Volume VIII, No. 2

35

FORTH Dimensions
NOW AVAILABLE

DESIGNING AND PROGRAMMING PERSONAL EXPERT SYSTEMS

CARL TOWNSEND AND DENNIS FEUCHT

Hands-on guidance in techniques that can be used to develop an individualized or expert knowledge system using Forth.

$19.00

FROM THE FORTH INTEREST GROUP

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