FORTH DIMENSIONS

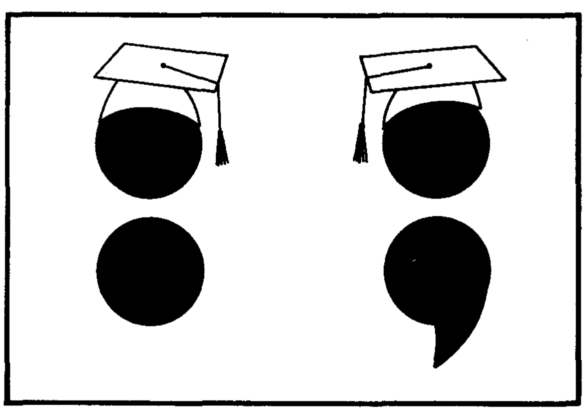
VOLUME IV, NUMBER 6 $2.50

INSIDE:

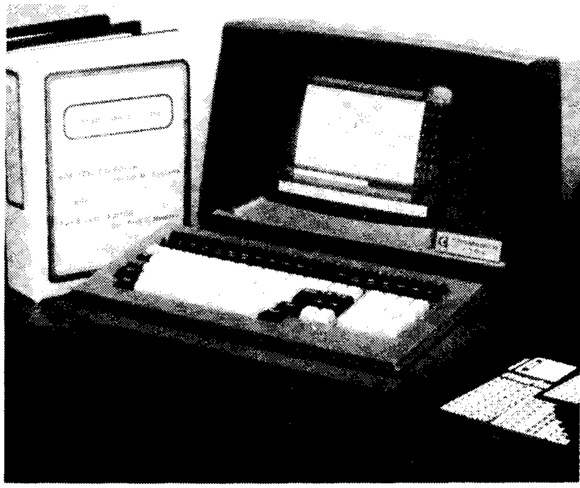
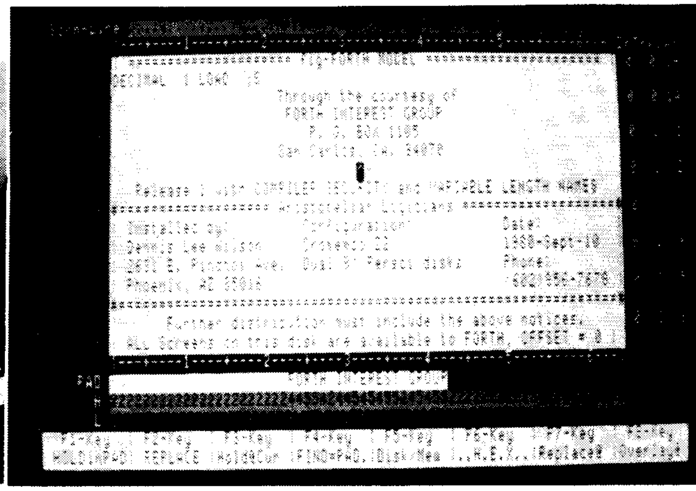
TEACHING FORTH

FEATURES:

DEPARTMENTS:



TEACHING FORTH

8080/Z80 FIG-FORTH for CP/M & CDOS systems
FULL-SCREEN EDITOR for DISK & MEMORY

$50 saves you keying the FIG FORTH model and many published FIG FORTH screens onto diskette and debugging them. You receive TWO diskettes (see below for formats available). The first disk is readable by Digital Research CP/M or Cromemco CDOS and contains 8080 source I keyed from the published listings of the FORTH INTEREST GROUP (FIG) plus a translated, enhanced version in ZILOG Z80 mnemonics. This disk also contains executable FORTH.COM files for Z80 & 8080 processors and a special one for Cromemco 3102 terminals.

The 2nd disk contains FORTH readable screens including an extensive FULL-SCREEN EDITOR FOR DISK & MEMORY. This editor is a powerful FORTH software development tool featuring detailed terminal profile descriptions with full cursor function, full and partial LINE-HOLD LINE-REPLACE and LINE-OVERLAY functions plus line insert/delete, character insert/delete, HEX character display/update and drive-track-sector display. The EDITOR may also be used to VIEW AND MODIFY MEMORY (a feature not available on any other full screen editor we know of.) This disk also has formatted memory and I/O port dump words and many items published in FORTH DIMENSIONS, including a FORTH TRACE utility, a model data base handler, an 8080 ASSEMBLER and a recursive decompiler.

The disks are packaged in a ring binder along with a complete listing of the FULL-SCREEN EDITOR and a copy of the FIG-FORTH INSTALLATION MANUAL (the language model of FIG-FORTH, a complete glossary, memory map, installation instructions and the FIG line editor listing and instructions).

This entire work is placed in the public domain in the manner and spirit of the work upon which it is based. Copies may be distributed when proper notices are included.

	USA	Foreign AIR
☐ **FIG-FORTH & Full Screen EDITOR package**		
Minimum system requirements: 80x24 video screen w/ cursor addressability 8080 or Z80 or compatible cpu CP/M or compatible operating system w/ 32K or more user RAM Select disk format below, (soft sectored only)	$50	$65

☐ 8" SSSD for CP/M (Single Side, Single Density)

Cromemco CDOS formats, Single Side, S/D Density

☐ 8" SSSD ☐ 8" SSDD ☐ 5¼" SSSD ☐ 5¼" SSDD

Cromemco CDOS formats, Double Side, S/D Density

☐ 8" DSSD ☐ 8" DSDD ☐ 5¼" DSSD ☐ 5¼" DSDD

Other formats are being considered, tell us your needs.

	USA	Foreign AIR
☐ Printed Z80 Assembly listing w/ xref (Zilog mnemonics)	$15	$18
☐ Printed 8080 Assembly listing	$15	$18

TOTAL $______ ______

Price includes postage. No purchase orders without check. Arizona residents add sales tax. Make check or money order in US Funds on US bank, payable to:

Dennis Wilson c/o
Aristotelian Logicians
2631 East Pinchot Avenue
Phoenix, AZ 85016
(602) 956-7678

Letters ....

Yes on Operating Systems

Dear FIG,

On the subject of operating systems and FORTH, I must admit that I much prefer Dr. Delwood's or Mr. Arkley's approach of using the operating system that you are currently running in, rather than creating an entire new one in FORTH. They are right—operating systems are designed to manipulate files; let them! This solves several problems that have arisen in that FORTH would no longer need to have an entire and separate disk all to itself, nor would it necessarily be cut off from the wealth of utilities that have been developed in almost any operating system. Why should the user have to worry about (and remember!) which screens contain which things. Everyone I know has a sequence of blocks off somewhere that are set aside for a listing and description of what is where on this disk. What is the difference between having to list that block and just typing "DIR" or "CATALOG?"

This is already being done to some extent. There are versions of FORTH that set aside a file under CP/M that contains all the screens, thus allowing FORTH to co-exist on a standard CP/M disk (though not to interact with other things on that disk), versions of FORTH that contain screens for saving and retrieving files under the operating system of that particular machine, etc.

The lack of a block-oriented file is, in my opinion, not a big problem. The standard block structure is nice for many things, but a real pain for others. There is no reason that it could not co-exist peacefully with the operating system's file structure. Then you could use whichever one best suited your needs at the time. Applications are created as standard DOS files, which are read into a buffer area for editing. One of the primary benefits of this is that you are not limited to an artificial 1K range for a definition, so you can spread it out as much as you like, indenting and commenting to your heart's content. The resulting readability lays to rest forever the argument that FORTH code is unreadable to the uninitiated. This buffer is flexible and moveable, starting above the dictionary (like **PAD**), and ending just below DOS itself (or wherever you tell it to end).

In William Graves' FORTH II for the Apple II, programs are saved through DOS with the command **DWRITE** and returned to the buffer with the command **DREAD**. You compile an application with the command **LOAD** (nothing on the stack), which **LOAD**s the entire buffer. Alternately, you can load directly from the disk with the command **DLOAD**. Standard DOS commands are prefaced with the word DOS: (i.e. **DOS: CATALOG**). What could be simpler? You can insert a file into another file, thus allowing you to save small files of standard applications (as you now do screens) that would be useful in many programs, or you can **DLOAD** a file directly from another file (a file can even **DLOAD** itself!?!).

These structures do not sully or defile FORTH, or turn it into something it Was Never Meant To Be. Rather, these file systems are just another example of the tremendous versatility and extensibility of FORTH.

I think it would not be out of line for the Standards Committee to look into adding words like those above to a future release of FORTH, not as a replacement to blocks, but as an alternative. It would make FORTH very little less *dependent* than it now is—just as certain words (**KEY, EMIT, ?TERMINAL**, etc.) must change from installation to installation, so to would these words need to change.

FORTH will never really gain acceptance in the Real World if you have to tell people they have a choice: they can either use FORTH with their hard disk, or everything else in the world.

Please keep up the excellent work! The only complaint I have is that *FORTH Dimensions* comes out *far* too infrequently to suit me!

Nick Francesco
Rochester, New York

While I personally prefer "native FORTH" directly controlling disk and other systems, I agree that running under an operating system can be useful. I definitely agree to the need for standardization, but this is unlikely to come from the Standards Team for a good long while. Just to clear up what sounds misleading, a FORTH that runs as its own operating system does not run in some other operating system, it replaces it. Similarly it doesn't need "an entire and separate disk all to itself." —Editor

More!

Dear FIG,

After having received volume IV of *FORTH Dimensions* we would like to purchase the three previous volumes. This is an excellent publication!

Richard Beers
Alpha Computer Services
Virgin Islands

No Point in Fixed Thinking

Dear FIG,

There exists a hardware trend which is possibly making the "Philosophy of Fixed Point" obsolete.

I recently purchased from FORTH, Inc.; Polyforth 2 with 8087 support for my IBM PC. Polyforth uses the 8087 register stack as an extra FORTH stack. I believe that this efficient use of the 8087 architecture makes FORTH an unbeatable number

Continued

FORTH Dimensions

Published by FORTH Interest Group
Volume IV, No. 6
March/April 1983
Editorial
Leo Brodie
Publisher
Roy C. Martens
Typesetting/Production
LARC Computing, Inc.

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

Subscription to FORTH Dimensions is free with membership in the FORTH Interest Group at $15.00 per year ($27.00 foreign air). For membership, change of address and/or to submit material, the address is: FORTH Interest Group, P.O. Box 1105, San Carlos, CA 94070.

Letters ... (Continued)

cruncher. Consider the following execution times that I measured.

SWAP	23 microsec.
*****	47 microsec.
FSWAP	14 microsec
F*	24 microsec

The floating point numbers are 64 bit (80 internal to the 8087). Besides being fast, by storing floating point numbers on a stack that is separate from the stack on which integers are stored, a major programming inconvenience is avoided, i.e. there are no operations between the two numerical types so putting both on the same stack requires excessive stack manipulation.

Clearly, in view of the above results, a fixed point philosophy is already completely inappropriate with an 8087 equipped machine. Now the question arises whether in ten years or so most microcomputers will be equipped with similar numeric processors. If so, and no floating point FORTH standards are in vogue at that time, FORTH will remain an unpopular number crunching language in spite of its potential superiority in this regard. I therefore recommend that you fixed point philosophers revise your thinking.

Steven A. Ruzinsky
Cicero, Illinois

Potpourri

Dear FIG,

First off, I must compliment Leo Brodie on his superb QTF application (IV/3 and IV/4) upon which I am composing this!!! Those new members who haven't got those back issues and don't possess a word processor *must* find them... Your life may depend on it if you ever have to write anything someday. The week it took to translate Brodie's code into FORTH-77 for my homebrew word-addressing machine was damn well worth it!

Don Colburn's letter (IV/4) was excellent. One thing he missed, tho. As I have pointed out in Guy Kelly's group (The San Diego FIG Chapter), people who promulgate "standards" should call them "alleged standards," as in "IEEE-488 Alleged Standard." The existence of FORTH standards, ala FORTH-79 or FORTH-83 does not necessarily mean compatibility. That would be nice, but it doesn't stand up in the real world. As EFUG pointed out to the Standards Committee, people *do* (but apparently not in the U.S.) implement FORTH on other than machines that address 8-bit bytes. A casual glance at FORTH-79 or the draft FORTH-83, however, shows that a "Standard System" is effectively impossible on a machine that has other than 8-bit addressing granularity! The FORTH community must realize that "standards" should serve primarily as a medium of exchange. The very nature of FORTH and its applications dictate this. This has always been a sore point with me. When I was exposed to FORTH initially, it was a word-addressing system. I liked it better that way.

Laxen's article on choosing names ...Both he and Brodie missed an important point. This is especially true if you look at the source for the QTF Editor where flags are being set and cleared all over the place. Beyond:

```
1 CONSTANT T(RUE)
0 CONSTANT F(ALSE)
```

They missed:

```
: ON    T SWAP ! ;
: OFF F    SWAP ! ;
```

Which gives:

```
: ESCAPE    ?ESC ON ;
```

Which makes much more sense than:

```
: ESCAPE    T ?ESC ! ;
```

Early in 1982 I needed a microassembler for a project at work. Not having seen Mr. Cholmondeley's work (III/4), I repeated a lot of it. My experience, however, indicates that a FORTH-based microassembler is the only thing that makes sense for this sort of work if you haven't got AMD's System 29 or such. I had to write about 256 x 40 of microcode and had access to a Z-80 CP/M system and 8080 fig-FORTH. Despite some hassels over having to say **SMUDGE** after **CREATE**, I was able to produce assembly listings and PROM dumps in Intellec format for the PROM blaster in about 24 hours work—which included coding up the microassembler from scratch! The performance brought frowns of dismay from the PASCAL- and C-freaks around the shop.

Glenn A. Toennes
DECOM Systems, Inc.
San Marcos, California

Thanks for your thoughts. Regarding the flag setting words, Laxen specifically recommended your proposed definitions in his article, except he called them **SET** *and* **RESET**. *As for me, in this particular application I find* **T ?ESC !** *to be quite readable, and also more symetric with* **?ESC @**. *The real problem is FORTH's use of @ and* **!**. *—Editor*

Kansas Canvass

Dear FIG,

I would like to locate some FIG members in the Eastern Kansas area. According to your FIG chapter listing from Vol. IV, No. 4 there is only one chapter in Kansas and it is a special interest group (Nova Group). If you would publish this letter maybe a few members in my area would like to start a chapter. It sure would be nice to talk with someone else interested in FORTH. As far as I know I'm the only FIG member east of Wichita.

Also Vol. IV, No. 4 was quite interesting. I received the issue just as I was developing a piece of telecommunications test gear in my job. Some of the ideas will save me at least 2K of ROM space. Keep up the good work.

E.J. McKernan IV
Datalog
Emporia, Kansas

FORTH as a Teaching Language

Albert S. Woodhull, Ph.D.
Hampshire College
Amherst, Massachusetts 01002

At Hampshire College we take a pause between the regular semesters for a "January Term." During January the expectation is that students and faculty will engage in intensive activities of the sort that are difficult to manage during the regular semester. This year, with help from my colleague Bill Marsh, I undertook to introduce a number of undergraduates to the FORTH language. I think FORTH has some features that could make it very useful as a teaching tool, but there are drawbacks as well. I will describe below the course and some of my conclusions regarding the possibilities for FORTH as a teaching language.

The course was called "Building a Small Compiler," and was publicized as being about the techniques used in implementation of high level languages. I proposed a goal of completing, as a cooperative project, a cross-compiler to allow high level development of programs to be run on smaller systems. The description listed familiarity with at least one computer language as a prerequisite.

A varied assortment of students appeared at the first class. The entry level programming courses at Hampshire have been taught in either APL or Pascal, and there was an expert programmer with little experience in the other language representing each. There were also a student with considerable experience in several languages, an intermediate Pascal student, and students who had learned FORTRAN and BASIC outside of Hampshire College. A few of the participants had some microcomputer assembly language experience; none had ever before used FORTH.

I started out with an exploration of the techniques used by microcomputer implementations of BASIC and Pascal in scanning for keywords and parsing expressions. The initial assignments were simple exercises simulating these processes, which students were expected to complete using the languages with which they were already most familiar. Individual presentations of solutions in each of these four languages gave me a chance to evaluate the starting levels of the students and provided them all with practice in explaining and translating algorithms.

We moved on fairly quickly to an examination of various ways of representing expressions, and the relationships between string and tree representations. These equivalent representations became a theme later, as I went into tracing the flow of control during program execution, but initially the goal was just to introduce postfix notation. At this point students were assigned a few exercises to familiarize themselves with the systems available, three S-100 CP/M systems of different hardware configurations. An expanded version of 8080 fig-FORTH with a screen editor designed for student use was available. All students had Brodie's *Starting FORTH* text, and after this orientation all showed themselves able to learn the language on their own, although some formal presentation of the control structures **BEGIN ... UNTIL** and **BEGIN ... WHILE ... REPEAT** was necessary for those without Pascal experience.

With the cross-compiler project as the focus I shifted the emphasis toward gaining an understanding of how FORTH works internally. For the less sophisticated students the necessary explanations of stacks and linked lists constituted an introduction to the general idea of data structures. It seemed to me there was a nice kind of symbiosis in the way the developing understanding of the linked list idea reinforced and was reinforced by the FORTH concept of vocabularies. I pushed this particularly, as one of the easier ways to a cross-compiling system is to change the order of the dictionary links.

The last part of the month was devoted mostly to investigation of how FORTH's inner interpreter worked. Exercises included writing, in FORTH, routines that could "decompile" the address lists into which FORTH definitions are compiled, and which could trace the flow of execution of FORTH words. These exercises were aimed at development of an understanding of how the FORTH "virtual machine" works and of what would be required to implement a FORTH system on a new target machine.

The three class weeks which constitute the January Term proved to be too short a time for carrying through on the cross-compiler project I originally proposed. I knew this was likely to be the case even before the course began, but I was not displeased with what did develop. In particular, I found it very interesting to gain a perspective on the usefulness of FORTH as an educational tool.

I have taught assembly language programming before, and I appreciate assembly language from a teacher's point of view as much for its importance in helping people to understand what the underlying machine is doing as for its usefulness in increasing speed or decreasing memory requirements. I think FORTH can serve a similar purpose in teaching computer scientists, but the goal can be reached more quickly, given that (as I find is usually true) the students already have done some high level programming. FORTH has some other educational advantages as well. Because it has a direct interpretation capability program modules can be debugged much more easily than in a "compile-only" language like Pascal. Yet FORTH is also a compiler and in a course for advanced students such as I have described FORTH can be used to gain an insight into the compilation process it-

Continued

FORTH as a Teaching Language (Continued)

self. With most high level languages the operation of the compiler or interpreter are not accessible to the user, nor can one count on similar principles of operation in different implementations. In FORTH it is very easy to follow the transformations that take place after a new definition is entered from the keyboard. Rules about declaration of variables take on new meaning when they can be shown to be necessary for the proper compilation of definitions that refer to the variables. Comparisons with other languages, particularly Pascal, can be made that point up clearly which rules are necessary because of machine restraints (i.e., variable declaration before use) and which rules go beyond what the machine requires for the sake of imposing discipline on the programmer (i.e., Pascal's requirement that all variables be declared at the beginning).

FORTH has its problems as a teaching language, as well. Its extreme flexibility makes it possible to do almost anything desired in FORTH, but this can cause confusion. As an example, it seemed essential for my purposes to be able to define a word recursively, and it is rather easy to do this by toggling the smudge bit in a dictionary header at compile time. I found the use of the [and] words, which allow for execution while a definition is being compiled, one of the most difficult things for students to grasp. Another feature of FORTH that seems to cause frequent errors is the difference in the behavior of **VARIABLE** and **CONSTANT** words, which return in one case the address of the variable and in the other the value of the constant.

My conclusions at this point are based on a very small sample of students in a specialized teaching situation, but I am very much interested in the possibilities of a language like FORTH for use as a teaching tool. I think at this point I would not offer a FORTH course as an introduction to programming, but I might change my opinion on this point if I had available a good package of utility words that would make it easy for students to do extensive numerical computations. I am seriously considering offering a course in FORTH as a second computer language. Particularly at a liberal arts college like Hampshire I could see a course of this sort serving a valuable role for science and other students who want to go farther than just learning to program, but who stop short of becoming computer science majors.

FORTH is above all a small system language, and teaching about computers has until now been based on large systems and the languages written for them. For example, as I mentioned above, APL and Pascal have been used, through time-sharing on the University of Massachusetts CYBER, for almost all the introductory language teaching at Hampshire College. The arrangements for this are under a great deal of strain, however. The microcomputer revolution has generated enormous pressure on the large system from students who want to learn about computers. As equipment becomes more affordable faculty members are obtaining their own computers for personal work and colleges are considering buying self-contained systems instead of buying or renting time-sharing terminals.

Most experienced teachers of computer languages agree that the BASIC which comes with small computers is not a good language for teaching programming. Some languages, like APL, do not adapt well to the microcomputer. Pascal seems to be making the transition, and in fact next year the introductory Pascal course at Hampshire College will use a new microcomputer laboratory across the hall from the old terminal room. Those of us who are familiar with the advantages of FORTH and its particular ability to make efficient use of limited machine resources ought to be giving some consideration to the question of how FORTH might fit into the college curriculum. Very little has been written on the use of FORTH as a teaching tool, and I would like to encourage others who have experience in teaching FORTH in undergraduate colleges to share their observations.

Albert S. Woodhull, Ph.D. is an Assistant Professor at the School of Natural Science, Hampshire College.

Teaching FORTH on a VAX

by Vernor Vinge
Department of Math Sciences
San Diego State University
San Diego, California 92182

During the Fall semester of 1982, I taught an introduction to FORTH at San Diego State University. In this article, I discuss the programming environment I devised and the resulting course.

The Hardware

Ideally, FORTH should be taught to students who each have their own FORTH system on their own computer—all in communication with the instructor's system. This ideal will be achievable in the future, but for the moment any FORTH course at SDSU will have to run on (very crowded) University equipment. If the ideal is impossible, it might seem that the best alternative would be to supply the class with a lab of FORTH systems. At SDSU we have about 50 Apples for instructional labs, but there is no way for the instructor's system to communicate with these machines. I regarded such communication as especially important in this course, since much of my software was untested. I decided to teach the course with the University's timesharing VAX. This machine communicates at 300 baud with terminals scattered across campus. (It can also be reached by phone from off campus.) It supports many courses in all departments and at all levels. When used in compatibility mode, the VAX is a fast and accurate emulator of a PDP-11 running the RSX-11 operating system. This made it easy to install an instructional FORTH on it.

The Software

For several years I have been using John S. James' implementation of fig-FORTH [5] on a small LSI-11 system. The nucleus of this (public domain) system is written in MACRO-11 assembler language, and can be reassembled to run standalone, under RT-11, or under RSX-11. Reassembling the nucleus is easy, though the only modification I had made in the past was to vector the definitions of words like **KEY, EMIT, QUIT, CREATE . . .**

Once the James system was in my VAX account it was easy to get it running there. If every student's file contained a copy of the nucleus, then this version could be used for instruction. There would be inconveniences, however: (1) the disk allocation for each student would be greater than is normally given to students, (2) the fig-FORTH dialect is different from that of my text, (3) system modifications would be difficult for me to make during the semester.

Therefore, I rewrote the nucleus to allow two sets of screens, one in a file called TEACHER.DAT and the other in STUDENT.DAT. TEACHER.DAT and the runnable nucleus, FORTH.EXE, reside in the instructor's account and can be read/run by the students. Each student has his own STUDENT.DAT. The names of these files are invisible to users. Screens 1 through 70 are really TEACHER.DAT and screens 71 through 140 are STUDENT.DAT. The system behaves like a standalone FORTH with two disk drives (but where one of the drives is read-only). By reassembling the nucleus, the instructor can change the characteristics of the system for everyone. By editing screens 1 through 70 (TEACHER.DAT) the instructor can deliver announcements and software to the students. In particular, the instructor can install a FORTH front end that will tailor the system to the dialect he wishes to teach. (My method of operation was a little more complicated: I did almost all program development on an LSI-11 system and then used a terminal emulator to upload debugged materials to TEACHER.DAT in my VAX account. This permitted me to use a screen editor and other fast-terminal tools. It also reduced the amount of time TEACHER.DAT was opened to the instructor; this was important during the semester, because the students are locked out of TEACHER.DAT when the instructor is messing around there.)

I decided early on that I wanted to use *Starting FORTH* [1] as my text. Let me sing some praises: Of all the introductory FORTH books I've seen, Brodie's is the only one that gets all the way through **CREATE . . . DOES>** and the compiler words. The explanations and examples are extraordinarily clear. Differences between dialects are carefully noted. Brodie skillfully treads the line between saying too much and becoming implementation specific, and not saying enough and becoming vacuous.

(And with the praise, some brickbats: The lack of an index is a continuing inconvenience. It is very difficult to discover the level of deferral of the system and user variables described on pages 236-240.)

Once I had decided to use *Starting FORTH*, the question was whether to follow its dialect or FORTH-79. Shortly before the semester began, I saw James' article [6] describing proposed changes in the Standard. These changes where almost all in the direction of the *Starting FORTH* dialect. So, with a more or less clear conscience, I decided to go with the dialect of the text.

I wrote twelve screens on TEACHER.DAT to redefine fig-FORTH words whose meanings are different in *Starting FORTH*, and to add words that exist in *Starting FORTH* but not in FIG. I also installed the "forgiving **FORGET**" [8], and defined a word **(ZAP)** to make unfindable many of the FIG words that don't exist in *Starting FORTH*. (I did not **ZAP** those utility words—such as **NFA**—which would exist in some form on any development system.) Writing this front end was routine, but only because of the information in Haydon [4] and Ting [12].

The next step was to install Daniel's version of the FORTH, Inc. line editor [3] (slightly revised for *Starting FORTH*) on TEACHER.DAT.

Continued

Teaching FORTH on a VAX (Continued)

Given this environment, a typical student work session begins with logging on the VAX and typing the command "FORTH." Once in FORTH, the student types **1 LOAD** and the *Starting FORTH* front end (together with the editor) is loaded. Thereafter the environment is almost identical to that described in Brodie's book. (There are big structural differences, however. The underlying architecture —things like vocabulary linkage and the layout of the bottom of the stack —is still fig-FORTH. These differences are rarely noticed in high level programming, and did not cause much confusion even when we got to the later chapters.)

Occasionally the VAX intrudes on the illusion that the system is single-user FORTH: When the VAX is heavily crowded, disk access becomes much slower than on a single-user system. If the student's program crashes his system, it's necessary to get out of FORTH, unlock STUDENT.DAT, come back, and repeat the **1 LOAD** step. (I could have reduced this inconvenience by making **ZAP** reversible, and by precompiling the front end.) The only PDP-11 specific problem I noticed was the necessity of keeping **HERE** even.

The Students

Some say FORTH can be taught as a first language to persons with no computer background. I believe that with a friendly front end, FORTH could compete with languages like Logo for elementary school applications. I understand that such projects are afoot [9]. However, if the goal is to learn FORTH to write serious applications, then the students must be fairly sophisticated. The prerequisite for my course was assembler language (CPU irrelevant). This was not because I used much assembler in the course, but because of the background which knowing assembler implies. Teaching FORTH in depth involves number systems, pointers, data structures, and binding times. Without previous exposure to such things, the average student would have a very hard time. (If I had been sufficiently expert as a FORTH teacher, and if the course had been three hours per week instead of two, it might have been possible to teach the same range of topics with somewhat weaker prerequisites.)

About 24 students attended regularly throughout the semester. Only two students had FORTH on their own systems. (They were welcome to do the assignments on those systems.) The general level of competence was high; I suspect that my requiring assembler language was responsible for this.

The Topics

The course was to consist of 15 lectures, each 100 minutes long. (This time there were only 14 lectures; one day was lost when the campus was unexpectedly closed.) It was easy to proceed directly through *Starting FORTH*. The first eight chapters were covered at the rate of one or two per lecture. (I skipped the details of fixed point scaling and mixed length arithmetic.)

The pace slowed dramatically when we reached Chapter 9, which took four lectures. I discussed vectored execution at length. New features were illustrated with improvements that the students could make to their systems. For instance, I showed how to develop the **LOCATE** facility (page 245), and how **QUIT** could be revectored to give diagnostics with each "OK."

The nesting and unnesting diagrams were discussed and elaborated on. One of the few places where I used assembler language was to demonstrate the implementation of these features in James' FORTH.

I did the FIG implementation of vocabularies in detail. In fact, I probably went too far with this, though it did give me a chance to discuss sealed vocabularies.

An added topic was a comparison of Direct Threaded, Indirect Threaded, Token Threaded, and Subroutine Threaded code. This led to a discussion of the hardware implementation of FORTH [10]. This lecture would probably have been unintelligible if the class didn't have an assembler language background.

Chapter 10 took two lectures. After going through the text material, I showed how FORTH could accomplish Pascal-like read and write statements. (I don't think the power of FORTH's I/O constructs is immediately obvious. By showing how easily standard I/O from another language could be accomplished in FORTH, I hoped to give the class a starting point for appreciating that power.) In addition to presenting the usual virtual memory approach to disk, I demonstrated that vectoring the I/O primitives could be used to make the disk look like a terminal device.

Chapter 11 took another two or three lectures. Both **CREATE ... DOES>** and the compiler words were covered in detail. I found the compiler words a tricky topic. (Things would have been a lot trickier if I had had to explain the "smart" versions of words like **."** and **'**. After teaching this course, I am definitely an opponent of "smart" words.)

I finished the semester with recursion, FORTH assemblers, job prospects, and a survey of further sources of information.

When I teach the course again, I expect to be able to cover more material. In addition to the (few) topics I skipped in *Starting FORTH*, I hope to cover compiler security, local variables [7], and a few ideas from metacompilation.

The Programs

Of course, programming is one of the most important parts of a language course. I made three formal assignments. The first was to write a formatted dumper word that showed both octal and ASCII representations of memory. This word was used throughout the rest of the course. The second project was to write a vocabulary for studying Conway's "Life" [2]. I supplied the design for this project. (I believe instructor-enforced designs are a good practice where the students don't have design experience.) The application included a two-dimensional wrap-around array for the universe of Life, words for display and time-stepping, and words for the definition and placement of new Life creatures in the universe. It was a good example of how FORTH can be extended to provide a special purpose language. (The *Infoworld* version of "Life in FORTH" [11] came out the week I made this assignment, and made an interesting contrast with what I was asking the class to do.) The third assignment was an anthology of short projects with defining and compiling words.

In addition to real programming, we had a 25 minute quiz at the end of each lecture. (I provided them with a *Starting FORTH* version of the

Continued

Teaching FORTH on a VAX (Continued)

FORTH-79 Handy Reference for these exams.) This many quizzes may seem excessive, but there were two good effects: We only met once a week and it was very easy for students to get behind. The quizzes helped them stay current. Furthermore, the quizzes amounted to short programming assignments (although of the "virtual" kind). Far more than with conventional languages, it is possible in FORTH to ask short questions whose answers do something significant. For instance, on an early quiz I asked the students to write words to translate Morse code into readable text. (I didn't ask that they handle the Morse code "dot.") This is a project that would involve a main program and various peipheral complications in conventional languages. In FORTH, it is trivial.

The Bottom Line

Fifteen of the 24 students became adequate to good FORTH programmers, and at least 20 learned a significant amount about FORTH. The time-shared approach is attractive, unless you or your students have plenty of standalone FORTH systems.

Acknowledgment

The discussions and presentations at the weekly meetings of the San Diego FIG have been very valuable to me.

Bibliography

1. L. Brodie, *Starting FORTH*, Prentice-Hall, 1981.
2. [Various authors], *Byte*, December 1978.
3. S. Daniel, "The FORTH, Inc., Line Editor," *FORTH Dimensions*, vol. 3, nr. 3, p. 80.
4. G. Haydon, *All about FORTH*, Mountain View Press, 1982.
5. J. James, *PDP-11 FORTH User's Guide*, January 1980. [This document is copyright, but the software it describes is public domain.]
6. J. James, "FORTH-83 Preview," *FORTH Dimensions*, vol. 4, nr. 1, p. 29.
7. R. Jekel, "Local Variables for FORTH," *1980 FORML Conference*, p. 59.
8. D. Kilbridge, "Forgiving FORGET," *FORTH Dimensions*, vol. 2, nr. 6, p. 154.
9. G. Stevenson, private communication, October 1982.
10. K. Suralis, "Principles for a CODE-less FORTH Machine," *1981 FORML Proceedings*, vol. 1, p. 5.
11. M. Swaine, "Life in FORTH: Simple Sample Program," *Infoworld*, 11 October 1982, p. 33.
12. C. Ting, *Systems Guide to fig-FORTH*, 1st ed., Offete Enterprises, 1981.

FORTH Standards Corner

Compilation Addresses and Parameter Fields

Robert L. Smith

At the last meeting of the FORTH Standards Team, probably no subject generated more heated debate than the questions relating to compilation addresses, parameter field addresses, and which one should be used by **EXECUTE** or returned from **FIND** and **'** (tick). In FORTH-79, **'** returned a parameter field address and **FIND** returned a compilation address. The main use for the compilation address was for use with **EXECUTE**, and possibly with **COMPILE**. The parameter field returned by **'** could only be used with constants and variables. The parameter field of a **DOES>** word could only be obtained from the execution of the word which was created by the **CREATE DOES>** pair. Under FORTH-79 there is no way to obtain the parameter field address of a word from its compilation address (or vice-versa). However, many implementations of FORTH use simple indirect threaded code in which the parameter field address is two bytes greater than the compilation address. In such systems the user may use only the parameter field, for example, and the system converts internally to compilation addresses when needed. This presents a certain simplicity to the user. This scheme is one form of "mono-addressing," and was provisionally accepted by the Standards Team at the Washington D.C. meeting. This was incorporated in the first draft (A) of the proposed FORTH-83 Standard.

After distribution of Draft A, a number of people and groups objected strongly to the mono-addressing scheme presented. Many objections were the result of implementation problems. The compilation address is more fundamental than the parameter field address, and in many implementations it is very easy to convert a compilation address to a parameter field address while the inverse is very difficult. Even with indirect threaded code, some of the new approaches put the compilation addresses (usually with the heads) in one "address space" and the parameters in another address space. "Token code" systems generally completely separate the compilation addresses and the parameter fields. Direct threaded code systems, directly compiled and "JSR" systems may have a variable separation between the two addresses. Indeed, for some classes of words in some systems the parameter field address has no meaning at all.

Within the context of the rest of the proposed standard the main use of the parameter field (as determined from, say, **'** or **FIND**) is as a secondary reference to **DOES>** words. The subteam on "addressing" took into account the above facts and objections and suggested to the team as a whole that the fundamental address is the compilation address. This is the address to be returned by **FIND**, **'**, and **[']**. This is the address to be used by **EXECUTE**. When the parameter field is needed, a conversion word named **BODY** is to be used. This performs the same action as the fig-FORTH word **PFA**. After much debate the team decided to accept the recommendation of the subteam. Under the proposed standard the programmer has access only to the parameter field of variables and **DOES>** type words. The use of **BODY** for any other type of word does not seem to be useful as part of a Standard Program.

Editor's Note:
Robert Smith is the current Secretary of the FORTH Standards Team and was a member of the original FORTH Implementation Team for FIG. He is employed by ESL Inc. in Sunnyvale, California.

Z-80® and 8086 FORTH

PC/FORTH™ for IBM® Personal Computer available now!

FORTH Application Development Systems include interpreter/compiler with virtual memory management, assembler, full screen editor, decompiler, demonstration programs, utilities, and 130 page manual. Standard random access disk files used for screen storage. Extensions provided for access to all operating system functions.

Z-80 FORTH for CP/M® 2.2 or MP/M	$ 50.00
8086 FORTH for CP/M-86	$100.00
PC/FORTH for IBM Personal Computer	$100.00

Extension Packages for FORTH systems

Software floating point	$100.00
Intel 8087 support (PC/FORTH, 8086 FORTH only)	$100.00
AMD 9511 support (Z-80, 8086 FORTH only)	$100.00
Color graphics (PC/FORTH only)	$100.00
Data base management	$200.00
Symbolic Interactive Debugger (PC/FORTH only)	$100.00
Cross Reference Utility	$ 25.00
Curry FORTH Programming Aids	$150.00
PC/GEN™ (custom character sets, IBM PC only)	$ 50.00

Nautilus Cross-Compiler allows you to expand or modify the FORTH nucleus, recompile on a host computer for a different target computer, generate headerless code, and generate ROMable code with initialized variables. Supports forward referencing to any word or label. Produces load map, list of unresolved symbols, and executable image in RAM or disk file. No license fee for applications created with the Cross-Compiler! Prerequisite: one of the application development systems above for your host computer.

Hosts: Z-80 (CP/M 2.2 or MP/M), 8086/88 (CP/M-86), IBM PC (PC/DOS or CP/M-86)
Targets: Z-80, 8080, 8086/88, IBM PC, 6502, LSI-11, 68000, 1802, Z-8

Cross-Compiler for one host and one target	$300.00
Each additional target	$100.00

AUGUSTA™ from Computer Linguistics, for CP/M 2.2	$ 90.00
LEARNING FORTH, by Laxen & Harris, for CP/M	$ 95.00
Z-80 Machine Tests Memory, disk, console, and printer tests with all source code in standard Zilog mnemonics	$ 50.00

All software distributed on eight inch single density soft sectored diskettes, except PC/FORTH on 5¼ inch soft sectored single sided double density diskettes. Micropolis and North Star disk formats available at $10.00 additional charge.

Prices include shipping by UPS or first class mail within USA and Canada. Overseas orders add US$10.00 per package for air mail. California residents add appropriate sales tax. Purchase orders accepted at our discretion. No credit card orders.

Laboratory Microsystems, Inc.

4147 Beethoven Street
Los Angeles, CA 90066
(213) 306-7412

Z-80 is a registered trademark of Zilog, Inc.
CP/M is a registered trademark of Digital Research, Inc.
IBM is a registered trademark of International Business Machines Corp.

Augusta is a trademark of Computer Linguistics
PC/FORTH and PC/GEN are trademarks of Laboratory Microsystems

Algebraic Expression Evaluation in FORTH

Michael Stolowitz

Editor's Note:
The approach that Mr. Stolowitz takes here resembles that used by Charles Moore in his BASIC compiler (FORTH Dimensions III/6, p. 175), although the two approaches were developed independently. We asked Mr. Stolowitz to refer back to Mr. Moore's implementation for comparison. He notes that Mr. Moore's approach is somewhat simpler since his algebraic parser only had to work in compile mode, and operators could be maintained on the data stack. Mr. Stolowitz's parser runs interpretively as well; therefore it requires an additional "operator stack."

Description of the Algorithm

Almost all systems which handle algebraic expressions do so by first converting the infix notation to postfix notation for evaluation. Since postfix or RPN arithmetic is built into FORTH, only a process for conversion of the notations is required. Before jumping into the algorithm in detail, let's consider an example:

A + B – C * (D / A) =

The algebraic rules tell us that the expression in the parentheses must be evaluated before the multiplication may be performed. The multiplication must precede the subtraction. The addition and subtraction operators have the same "precedence" so that the algebraic result is independent of the order in which they are performed. On real machines with round off errors, it is probably best to execute them in some consistent fashion in order to obtain reproducible results. We will use left to right evaluation in the following discussion. Let's now examine an RPN statement of the same expression:

A B + C D A / * –

There are several significant items to note. First, the operands appear in exactly the same order in both the infix and postfix versions of the expression. As the operands are encountered in a left to right scan of the expression, each will push its value onto the data stack. The next thing to note is that the appearance of each of the operators from the algebraic expression has been delayed until a point in the RPN expression where all of the required operands will be available on the stack. This allows the operators to be immediate, i.e. they execute as soon as they are encountered in the RPN expression. The final point to note is that it is possible to obtain an RPN expression with the above properties without the use of parentheses.

The algorithm for infix to postfix conversion uses an additional stack on which to hold operators while they are being delayed as described above. The algorithm is simply to place each operator on the operator stack as it is encountered in a left to right scan of the algebraic expression; however, an operator may not be pushed on top of another which has a precedence equal to or greater than its own. If necessary, operators are removed until the new one may be pushed. Operators removed from the stack are output by the algorithm. Since there is no change in their sequence, the algorithm passes operands directly to the output.

When a **(** is encountered, a special operator called a floor is placed on the stack. The floor, while having a very low precedence, may be placed on top of whatever is already there. The floor will allow additional low precedence operators to be pushed. The effect of a **)** is to dump the operator stack down to and including the most recent floor.

Unless the entire expression had been enclosed in parentheses, there will be one or more operators remaining on the stack when the end of the expression is reached. This is because there is no way of anticipating the presence or absence of additional operators. A signal is required (the = key on an algebraic calculator) to dump the balance of the operators from the stack completing the translation.

Let's evaluate the expression in the example using this algorithm. The **A** operand would pass through to the data stack. The **+** would be pushed onto the operator stack since there is no operator there with a higher precedence. The **B** operand would then go to the data stack. At this point the **–** is encountered which has the same precedence as the **+** which is already on the stack. The **+** is removed and executed leaving **A + B** on the data stack and the **–** is pushed onto the operator stack. The **C** goes to the data stack. The ***** has a higher precedence than the **–** so it may be pushed onto the operator stack. Now we come to the **(** so a floor is placed on top of the **–** and the *****. The **D** goes to the data stack and the **/** is pushed on top of the floor. The final **A** goes to the data stack. Now comes the **)** which causes the **/** to be dumped and executed and the floor to be removed. The **/** produces the intermediate result **D / A** on the top of the data stack. At this point the **=** causes the rest of the operator stack to be flushed. The ***** and **+** are executed in the sequence in which they are removed producing the same execution sequence as the RPN expression.

It should be clear from the above example that the complexity of the expressions which may be evaluated is limited only by the depths of the operator and data stacks.

While the preceding discussion refers to the "evaluation" of an expression, the actual calculation need not be performed at the same time as the translation to RPN. The resulting RPN sequence of operators and operands could be recorded for later execution. This process is, of course, "compilation" and permits rapid evaluation of the expression for different com-

Continued

Algebraic Expression Evaluation in FORTH (Continued)

```
SCR # 66
\ ALGEBRAIC                                               30NOV82MCS
CREATE OP   44 ALLOT
: ?INTERP ( pfa -- ) CFA STATE @  IF , ELSE EXECUTE THEN ;
: OPP@    ( -- addr ) OP DUP @ + ;
: >OP     ( pfa lev -- ) 4 OP +!  OPP@ 2! ;
: OP>     ( -- ) OPP@ 2@  -4 OP +!  DROP  ?INTERP ;
: LEV?    ( -- lev ) OPP@  @ ;
: ]A      BEGIN LEV?  WHILE OP>  REPEAT
          [COMPILE] FORTH ; IMMEDIATE
-->
```

```
SCR # 67
\ ALGEBRAIC                                               30NOV82MCS
: INFIX  ( lev -- )  ( old rpn op   new infix op )
   '  CREATE   SWAP   , ,   IMMEDIATE
      DOES>   2@ BEGIN  DUP  LEV? >  NOT  WHILE
         >R  >R    OP>    R> R>   REPEAT    >OP ;
VOCABULARY ALGEBRAIC IMMEDIATE   ALGEBRAIC DEFINITIONS
7 INFIX * *   7 INFIX / /
6 INFIX + +   6 INFIX - -
5 INFIX > >   5 INFIX < <   5 INFIX = =
4 INFIX NOT NOT
3 INFIX AND AND
2 INFIX OR OR
-->
```

```
SCR # 68
\ ALGEBRAIC                                               30NOV82MCS
: (     ['] CR   1    >OP ; IMMEDIATE
: )     FORTH   BEGIN  1 LEV?  <  WHILE   OP>   REPEAT
        L  LEV? =  IF  -4 OP +!
                   ELSE  1 ABORT" Missing (" THEN ; IMMEDIATE
FORTH DEFINITIONS
: A[    0 OP !   [COMPILE] ALGEBRAIC ; IMMEDIATE          EXIT
Examples:     A[  A + B - C * ( D / A )  ]A
   or  : EPXR  A[  A + B - C * ( D / A )  ]A  ;
OBWRO'CONY ANGIELSKI is Polish for "Reverse English"!
```

Continued

Algebraic Expression Evaluation in FORTH (Continued)

binations of variables without returning to the original expression.

The Implementation

An implementation of the above algorithm is given in the appendix. The program begins with the definition of a data structure, in this case an operator stack. The word **OP** is defined to return the address of a block of RAM allocated for this purpose.

The stack is used to store double word (four byte) entities consisting of operator/precedence pairs. The double word at the base of the stack is used for the top of stack pointer and contains the byte offset from the base of the stack to the top element, a zero value indicating stack empty. The other half of the pointer double word contains a precedence of zero.

The word **>OP**, pronounced "to op" is the stack push operation. It expects an operator and precedence level on the data stack, bumps the operator stack pointer by four and stores the double word at the new TOS location. **>OP** uses a primitive **OPP@** ("op pointer fetch") to obtain the absolute location of TOS in memory.

OP> pronounced "from op" is the stack pop operation. The precedence level is discarded at this point since it is not required in RPN. **OP>** is "state smart" because it uses the conditional interpretation word **?INTERP** which looks at the system variable STATE to determine if the system is currently interpreting or compiling. If compiling, it compiles the operator's execution address in the dictionary. If interpreting, it executes the operator immediately.

The word **INFIX** does the bulk of the work. **INFIX** is given a precedence level, the name of an existing RPN operator and the name to be given to the new corresponding infix operator. It looks up the execution address of the existing word and then creates the new word placing both the execution address and precedence level in its parameter field. When any of the words created by **INFIX** are invoked, the code following the word **DOES>** will be executed using the execution address and precedence from the particular words parameter field as arguments. This code implements the algorithm described previously. The execution address and precedence for the operator are pushed onto the operator stack with operators of lower precedence being popped first. The operators created by **INFIX** are kept in a separate vocabulary so that they might have names which would otherwise conflict with standard FORTH words. This vocabulary has been named ALGEBRAIC.

The definitions for **(** and **)** are also segregated from the FORTH dictionary because these delimiters are commonly used for comments. The new **(** simply pushes a dummy operator with a precedence of 1 (to distinguish it from an empty stack which has a precedence of 0) onto the operator stack. The **)** dumps the stack until it finds the level 1 operator or issues an error message if there is not one to be found.

The final two definitions are placed in the FORTH vocabulary. The first is **A[** which is used to enter algebraic mode. It selects the ALGEBRAIC vocabulary and clear the operator stack. The second word is **]A**. This word is used to exit algebraic and to reselect the FORTH vocabulary, but first, it performs the "end of expression" function by dumping any operators remaining on the operator stack.

Two factors allow all of the above code to work in either execution or compilation modes. First, the words created by **INFIX** and the parentheses operators are all **IMMEDIATE** words, meaning that they execute even though the system might be compiling (like compiler directives in other systems). Thus all of the operator stack activity will occur as the expression is scanned independent of the state of the machine. **?INTERP** will take care of the state difference as the operators come off of the stack by compiling or executing as appropriate.

The other factor allowing state independence is that the operands used are all self fetching at run time. Oper-

Continued

Algebraic Expression Evaluation in FORTH (Continued)

ands of this type include literals and constants. Both may be compiled and return values when executed. This is not a restriction as constants may be conveniently used as variables. There are many references in the literature on **TO VARIABLES**.

A final significant observation must be made in regard to the operands. While it was convenient to think of the operands as sixteen bit integers since this is what most FORTH systems have arithmetic operators for, in no way does any of the above code depend on that fact. The operands could just as well have been double words or floating point words which fetched themselves to a floating point stack in an arithmetic chip. The only requirements are that the operands be self fetching to some stack and that the RPN operators be appropriate for the data types and stack used. The generality of the above code and the range of applications possible through changing only those words created by **INFIX** takes some time to appreciate.

In summary, a relatively minor extension to FORTH's compiler has been presented for the compilation of algebraic expressions. A mechanism is provided for the definition of infix operators in terms of their RPN equivalents and a precedence. The resulting code consists entirely of RPN operators previously existing in the system so that none of the compiler extensions are required for execution. The techniques are easily extended to include additional operations or data types. While this system provides a convenient tool for many kinds of applications, the user is cautioned against permanently isolating the natural arithmetic of his machine. After all, algebraic has been described as OBWRO'CONY ANGIELSKI which is of course Polish for "Reverse English."

Michael Stolowitz is a Consulting Engineer based in Danville, California.

Meta Compiling I

Henry Laxen

Meta Compiling is an often heard term in FORTH, and yet most people approach it with fear and anxiety. This is unfortunate since it is really not that difficult and it is extremely powerful. Many purposes have been attributed to Meta Compiling, such as generating new FORTH systems, creating a custom application, cross compiling code for a different target machine, removing the names (headers) from the code, and generating ROMable code. All of these are different benefits of the Meta Compiling process, but they may or may not be the only way to accomplish the task. For example FIG allowed people to create new FORTH systems by supplying assembly language listings of FORTH which people could enter into their computer and assemble with their assembler. No Meta Compiler ever entered the picture. Before exploring Meta Compiling in detail, let's first look at the dictionary definition of the word META.

META a prefix meaning 1. changed, transposed [metamorphosis, metathesis]; 2. after, beyond, higher [metaphysics]

Meta Compiling in FORTH combines attributes of both of the above definitions. It occurs on a "higher level" from ordinary compiling and involves a change from one environment to another. In one sentence, Meta Compiling in FORTH is a process in which FORTH code is compiled in one environment and executed in another. The environment in which the code is compiled is called the HOST system. The environment in which the code compiled by the Meta Compiler will finally execute is called the TARGET system. One of the main difficulties encountered in Meta Compiling is the confusion that naturally arises out of the interactions between the different environments. Many words in the Meta Compiler have totally different meanings depending on the context in which they are used.

In the first part of this exposition we will look in detail at one of the central issues of Meta Compiling, namely that of storage allocation. We will leave the issue of context for the next article. Think of a Meta Compiler as a machine in which FORTH Source Code is cranked in and Target Object Code is cranked out. In any software project one of the main resource allocation problems is how to allocate memory. The same is true in Meta Compiling, and this article will address the issue of memory allocation for Meta Compilers. The problem then is to construct a mapping in which the Target Image can reside, and to find a convenient way of manipulating that Target Image. Instead of reinventing the wheel, let's do it the way FORTH does it. FORTH has a set of words that read and write memory, as well as allocate and initialize space in the dictionary. Presumably we will need the same functions in the Target Image. This difference is that while the ordinary FORTH words that read and write memory, namely @ and !, operate on addresses, our new read and write memory words will have to operate on Target addresses. What we need is a word which will map Target addresses into Host addresses. Let's call this word **THERE** and it must behave as follows:

target-address **THERE** host-address

Using **THERE**, we can define the read and write memory words as

```
: @-T   THERE @ ;
: !-T   THERE ! ;
```

We append the **-T** suffix to indicate that we are fetching and storing into Target address. We can define **C@-T** and **C!-T** in a similar way. Next we want to implement something analogous to a dictionary in the Target System. The amount of space that has been allocated in an ordinary FORTH system is held in a variable called **DP**.

We can analogously define a variable called **DP-T** to hold the amount of space allocated in the Target System. Armed with that definition we can define the dictionary words as follows:

```
: HERE-T   DP-V @ ;
: ALLOT-T   DP-V +! ;
: ,-T   HERE-T !-T 2 ALLOTT ;
```

(Why is there an ordinary @ in the definition of **HERE-T** and a **!-T** in the definition of **,-T**?) Why have we gone through such an elaborate ritual? Let's take a quick look at what we can do with these words. Perhaps you recall how FORTH Assemblers work. (If not wait for a future issue and I will discuss them in this column.) The main idea behind FORTH Assemblers is that you define a set of FORTH words whose names are op codes for your particular machine. When these words are executed they assemble their machine language binary op code into the dictionary along with whatever parameters are required. For example the jump instruction on the 8080 is a hex C3 followed by the 16 bit address of where to jump to. The **JMP** word in the FORTH Assembler is thus defined as:

```
: JMP   C3 C, , ;
```

The **C,** assembles the op code into the dictionary and the **,** assembles the address that must have been left on the stack. Notice that the compiled code is inline in the dictionary. Now, using the **-T** definitions we defined above, we can now assemble code which will execute from a different address than where it was assembled. The corresponding definition for jump would be:

```
: JMP   C3 C,-T ,-T ;
```

This would assemble the op code in the next available location in the Target System, not in the Host System. Furthermore, it will jump to the specified Target address when it is executed, not to the Host address. What we have done is turned a FORTH Assembler that can assemble inline code words into a cross assembler that can assemble code that will execute in an environment other than FORTH.

If the significance of what has just been discussed has escaped you, don't feel bad. It escaped me the first six times also. Don't be fooled by the simplicity of the implementation. The mere fact that we can assemble or compile code in a different memory area than the one we are executing out of is very powerful. It is one of the cornerstones of the Meta Compiling process.

It now only remains to define the mapping word **THERE**, which takes a Target address and returns a Host address. The simplest approach, if you have enough user memory, is to sim-

Continued

Meta Compiling I (Continued)

ply define **THERE** as a constant offset as follows:

```
20000 CONSTANT TARGET-OFFSET
: THERE   TARGET-OFFSET + ;
```

You can't get much simpler than that. However, there are times when memory is tight or the application program is just too large to fit. What do you do then? In most other programming languages you either give up or start the entire application over from scratch. We in FORTH have the luxury of redefining a few words and the rest of the application will never know the difference. Let's take a look at how we can provide a mapping from Target to Host addresses without taking up any room in the Host dictionary. The answer is of course to use **BLOCK** as a means of mapping memory addresses into disk addresses. Consider the following:

```
10 CONSTANT TARGET-BLOCK
: THERE   ( target-addr -- host-addr )
1024 /MOD  TARGET-BLOCK + BLOCK + ;
```

We first divide by 1024 bytes per block, and get back a quotient and a remainder. The quotient is the block number and the remainder is the byte index into that block. All we have to do is add in the beginning block number, **TARGET-BLOCK**, and call our friend **BLOCK** to perform the mapping of a block number into a buffer address. Finally, we add in the byte index into the returned address and we are done. Or are we? There are two bugs in the above code, as it relates to Meta Compiling. See if you can find what they are.

The first bug will probably not bite you, but when it does it will produce very dramatic results and it will be obvious how to fix it. The problem is that when dealing with addresses, you should be very careful what kind of arithmetic you perform. Addresses are unsigned quantities, while division and multiplication deal with signed quantities. The above code works fine as long as the Target address is less than 32K. As soon as it is larger, **/MOD** returns a signed quotient and remainder, and we will be passing **BLOCK** a very strange block number. I will leave it to you to rewrite **THERE** to avoid this 32K problem.

The second bug is far more subtle, and in fact does not lie in the word **THERE** at all. You don't discover this one until you have crashed many many times. Recall the definition of **@-T** and **!-T** was:

```
: @-T   THERE @ ;
: !-T   THERE ! ;
```

Well, 1023 out of 1024 times this will work just fine. You see if we call **THERE** with a Target address that is congruent to 1023 modulo 1024, then **THERE** will return the address of the last byte in a block buffer. Since @ and ! act on 2-byte, 16-bit entities, the wrong results will be read or written. Rule of usage for **THERE** is that it takes a Target Byte Address and returns a Host Byte Address. Only a single byte address is returned. There is no guarantee that Target Address + 1 maps into Host Address + 1. That is a false assumption on the user's part. Anyway, how do we fix it? It really is

Continued

1983 ROCHESTER FORTH APPLICATIONS CONFERENCE

June 7 through June 11, 1983
University of Rochester
Rochester, New York

- The third Rochester Forth Applications Conference will be sponsored by the Institute for Applied Forth Research, Inc., and hosted by the University of Rochester's Laboratory for Laser Energetics. This year's conference has a format similar to that of previous Rochester conferences, with invited speakers focusing on an area of particular interest. This year's topic is robotics, which embraces many areas, including mechanical and electrical engineering, vision, artificial intelligence, computer networking, and automated manufacturing. Six invited lecturers will address the use of Forth in factory automation, the design of a useful robotic control language, teaching robotics and Forth, and expert systems which provide direction to autonomous robots. The series will cover the evolution of Forth from telescope control through the precision multi-axis camera controllers used in Hollywood to state-of-the-art robotics and applied artificial intelligence research.
- In addition to one day of invited lecturers, there will be three days of oral presentations, posters, and demonstrations, as well as working groups on specific topics of interest. All sessions will be held in the Wilson Commons at the University of Rochester.
- The registration fee of $300. covers all sessions, activities, meals, and the 1983 Conference Proceedings. Students may register for $150. Attendees have the option of student dormitory housing, at the rate of $58 double and $80 single for the duration of the conference. A list of nearby motels is also available. However, those people staying on campus will find a car unnecessary, as there will be a shuttle to the airport and the train station. Campus parking permits are available for $.75/day. The University Health Service facilities are also available for a single fee of $4 at registration.
- For more information, please contact: Diane Ranocchia
  Institute for Applied Forth Research, Inc.
  70 Elmwood Avenue
  Rochester, New York 14611
  716-235-0168

REGISTRATION FORM
Please return by May 15, 1983

Name: ______________________ Telephone: ______________

Address: ____

____

Are you planning on giving a: 10-20 minute talk? ________
poster? ________
demonstration? ________

If you are, we should have your 200-word abstract by April 15.

Registration fee: $300 ($150 for students) ________

University housing: yes no
single $80, double $58 ________

To stay over the night of the 11th, add $15 ________

If you chose double, who is your roommate? ______________________
(Roomates will be assigned if not indicated.)

Amount enclosed: (Please make checks payable to the Rochester Forth Conference.) ________

There is a vegetarian meal option. Check if desired. ________

Mail registration to: Rochester Forth Conference
70 Elmwood Avenue
Rochester, New York 14611

Meta Compiling I (Continued)

quite simple; namely, we must construct @ and **!** out of **C@** and **C!**, which only operate on byte addresses, not word address.

At this point we get into a small mess because many microcomputers are "byte-swapped," meaning that for a 16-bit word, the low order 8-bit half is stored first. The 8080 and 6502 are prime examples of byte-swapping machines. The newer 68000 computer is an example of a nonbyte-swapping machine. Anyway, to construct @ and **!** out of **C@** and **C!** we must be aware of the byte-swapping. Let's suppose we are on a byte-swapped machine, and let's take a look at how to implement @. I will leave the implementation of **!** as an exercise. Consider:

```
: @-T  ( target-addr -- value )
   DUP C@-T  ( addr low )
   SWAP 1+ C@-T  ( low high )
   256 *  +  ( hilo ) ;
```

Notice that only **C@-T** is used, so our rule of useage is not violated. This is rather slow on most machines because of the multiply, but it will certainly work. What would be nicer is to define a **CODE** word, say **FLIP**, which exchanges the high and low halves of a 16-bit word. Then we could replace the **256 *** phrase with **FLIP** and it would be much faster. If the machine were not byte-swapped then we would place the **FLIP** or the **256 *** after the first **C@-T** instead of the second. See if you can implement **!-T** in an analogous way.

What we have really done is implement a disk resident virtual memory system. It turns out to be very useful in many applications, not just Meta Compilation. Any time you need a very large array that will not fit in memory, the same technique will work. Next time we will look deeper into the Meta Compiling process and address the issue of how to actually generate the Target Image Code, now that we have a place to put it. Until then, good luck and may the FORTH be with you.

Henry Laxen is Chief Software Engineer for Universal Research, 150 North Hill Drive, #10, Brisbane, CA 94005, specializing in the development of portable computers.

New Product Announcements

A FULL-SCREEN EDITOR for DISK and MEMORY PLUS 8080/Z80 fig-FORTH for CP/M and CDOS Systems

The full screen editor allows use of terminals that transmit and receive ESCAPE sequences. Reverse video, blinking and half-intensity are also used, but provisions are made for terminals lacking these features.

All control characters show as " " and any character higher than 7E HEX shows as " ". (You may change these defaults if you prefer.)

This editor also has the unique ability to VIEW AND MODIFY MEMORY using all the full screen edit functions, (except "LINE-INSERT" and "LINE-DELETE" which are disk related.) Using the "NEXT-PAGE," "PREV-PAGE" and "PAGE-EDIT" function keys you may skip or browse thru memory (including the running operating system) a full page at a time, and change any memory location to any value desired.

The system consisting of the full screen editor along with the fig-FORTH model and many additional fig-FORTH screens is delivered on TWO diskettes. The first disk, readable by CP/M or Cromemco CDOS, contains the 8080 FIG source listing and an enhanced version in ZILOG Z80 mnemonics, with FORTH.COM files for Z80 & 8080 processors and a special one for Cromemco 3102 terminals.

The second disk contains FORTH readable screens including the extensive full-screen editor for disk and memory. This disk also has I/O port dump and traditional formated memory dump words plus many items published in *FORTH Dimensions*, including a FORTH TRACE utility, a model database handler, an 8080 ASSEMBLER and a recursive decompiler.

This entire work is placed in the public domain in the manner and spirit of the work upon which it is based. Copies may be distributed when proper notices are included. The two disks and a copy of the fig-FORTH Installation Manual are $65.

Contact: *Dennis Wilson, Aristotelian Logicians, 2631 East Pinchot Avenue, Phoenix, AZ 85016 (602)956-7678.*

FORTH/COMM

FORTH/COMM is a software package of high level FORTH routines which transfer information to and from FORTH systems while insuring the integrity of data. The features included in the package are

- Pre-installed drivers for IBM/PC, Apple Comm Card, CP/M
- Two levels of capability:
  Level 1—Transmit FORTH text screens with handshaking
  Level 2—Transmit FORTH text or raw binary screens with error detection and retransmission
- Routines for both sending and receiving
- Programs will not "hang" during data transfer
- Dumb terminal program included
- Remote operation can be achieved with the dumb terminal program

These routines have proven to be very reliable under a wide variety of conditions. Extra care has been taken to minimize the number of control characters to avoid interference with operating system codes. For example, the check word and screen sequence number are sent as a string of 4 ASCII hex digits. Idle time is minimized by having the sending routine fetch the next screen to be transmitted while the receiving routine is storing the last screen on disk.

FORTH/COMM includes *all source code*, manual, and program disk postpaid for $75.

Specify one of the following disk formats:

(FORTH-79 and fig-FORTH with 1024-byte buffers)

- 8″ CP/M—FORTH screens and file of FORTH screens
- 5″ PC-DOS—FORTH screens and file of FORTH screens
- 5″ Apple 3.3—FORTH screens

Contact: *Ren Curry, Curry Associates, P.O. Box 11324, Palo Alto, CA 94306, (415)322-1463.*

Call For Papers

5th FORML Conference

November 23-25, 1983
Asilomar Conference Center
Pacific Grove, California, U.S.A.

The 5th Annual FORML (FORTH Modification Laboratory) Conference will be held November 23-25, 1983 at the Asilomar Conference Center, Pacific Grove, California, U.S.A. FORML is a forum for sharing and discussing new proposals intended to benefit FORTH. It is not intended for beginning or casual FORTH programmers.

Registration includes a comfortable room, meals, conference attendance and notebooks containing the papers submitted. Attendees who are also contributors to the conference will receive first priority. A limited number of guests (family or friends not attending the conference) can be accomodated.

Topics presented should focus on guiding the evolution of FORTH. Compliance to a standard FORTH is recommended but not required. The following list suggests topics for presentations:

Hardware FORTH Implementations;
FORTH engines, architecture, instruction sets, single chip integration, p.c. board processors, microprogramming, coprocessors, hardware stack add-ons, ROM-chip projects.

Large Address Space Environments:
32-bit addressing, segmentation, memory mapping, extended addressing, bank switching.

Multiprogramming Architectures:
Approaches, multiuser, multitasking, interrupt handling, memory management, task control, intertask communication and coordination, UNIX-like pipes.

Nucleus Variations:
Virtual machine, threaded code techniques, virtual execution, relocation.

Operating System Environments:
Native architectures, device drivers, guest of other operating systems, file system interfaces.

System Generation Techniques:
Defining words, compiler writing systems, metacompilation variations.

Program Development Methodologies:
Problem analysis techniques, design criteria, style, coding standards, tools, team programming, documentation, training, quality control and evaluation.

Applications:
Process control, real-time techniques, robotics, scientific calculations, matrix operations, simulation, business applications, file systems, relational data base systems, other languages.

Abstracts for presentations should be no longer than 100 words and must be received by September 6, 1983. To obtain registration information and an author's kit please write to:

FORML
P.O. Box 51351
Palo Alto, CA 94303
U.S.A.

Technotes

+BUF BUG

David Cromley
Cheyenne, Wyoming

I have found a bug in **FLUSH** *(fig-FORTH Rel 1, Nov. 1980)*.

I suggest for SCR#92 line 4:

[LIMIT / 1+] (No. of **BUFFS +1**)

The present definition fails when **PREV** points to the buffer after **USE**, and this **PREV** buffer has been updated.

For example:

#1	#2	#3	#4

USE PREV

Presently, **FLUSH** will call **BUFFER** 4 times. **USE** will be, for these four times: 1, 3, 4, and 1. Buffer 2 will be missed.

Yes, this is a design error. In the interest of trying to keep the most recently referenced block in RAM buffer, it may occasionally not be correctly **FLUSHED**. *The culprit is* **+BUF** *and it's used in* **BLOCK** *and* **BUFFER**. *Your solution is one of several now in use.*

Most vendors who follow the FIG Model have corrected this and several other problems (carry problems in **U*** *and* **U/**, *expanding buffer size to 1024 bytes and Y register range in* **ENCLOSE**). *These updates point out the advantage of software products with vendor support, additional testing and review. The self installed FIG system has had remarkable acceptance, but it cannot supplant the broader resources of commercial offerings. —Bill Ragsdale*

Bit Array and Manipulations

Timothy Huang

Bit manipulation is one of the most useful techniques in microcomputer programming, particularly for hardware controls and graphics, for which FORTH is especially good. However, when I looked back through all the *FORTH Dimensions* issues, I failed to find any such program published. Well, here is one that I wrote a couple of months ago.

The main program is in Screen 117 with comments a la Henry Laxen. You should not have any problem understanding or using it. The only supportive word is **2** , which is defined in high level in Screen 116. This word can be and should be re-written in low level, because it is quite useful and time critical.

The only comment that should be added to the defining word **BIT-ARRAY** is that at compiling time [time 2], if the <number_of_bits> cannot be evenly divided by 8, then it will reserve up to the next byte. For example:

27 BIT-ARRAY TEST

will reserve and initialize to zero 4 bytes (32 bits) **TEST** bit array, even though the last 5 bits in the array's last byte are insignificant. Also, at execution time [time 3], there is no range check. It is supposed to be the programmer's responsibility to know what he/she is doing.

Screen 118 provides four bit manipulation words. The only comment that I would like to add here is the name of these words. **BIT-ON , BIT-OFF** , and **BIT-TOGGLE** are perhaps not as good as **ON , OFF** , and **TOGGLE**. You really should read Mr. Henry Laxen's article (*FORTH Dimensions*, Vol. 4/4). Good naming principles are an essential part of good FORTH.

Continued

```
SCR # 116                                                    TDH25OCT82

\  2^

: 2^   (n --- 2^n)    \raise to 2's n-th power
       DUP  0=  IF DROP 1
                ELSE 1 SWAP 0 DO  2* LOOP    THEN ;
;S
(P 2^ will yield a number which is the n-th power of 2.)


SCR # 117

\ BIT-ARRAY                                                  TDH25OCT82

: BIT-ARRAY    \defining word
      <BUILDS (number of bits --- ; reserved and initial.)
              8  /MOD  SWAP  IF 1 ELSE 0 THEN +
              HERE OVER  ERASE  ALLOT
      DOES>   (index --- #.of.bit.offset addr )
              SWAP  8  /MOD  ROT +   ;
;S
(P BIT-ARRAY defines a family of bit arrays. At the bit
   array creation time [time 2], it takes the form:
        number.of.bit.wanted  BIT-ARRAY  <name>
   At the execution time [ use the <name> bit array;
   time 3 ], it takes the form:
        index <name>  [index --- #.of.bit.offset  addr ])

SCR #118

\ BIT-ON BIT-OFF BIT? BIT-TOGGLE                             TDH25OCT82
: BIT? ( index <name> --- 1/0 )    \ 1 = on   0 = off
        C@  SWAP  2^  AND   0=   0=  ;

: BIT-ON ( index <name> --- )       \ set index bit on
        DUP  >R  C@  SWAP  2^  OR  R>  C!  ;

: BIT-OFF ( index <name> --- )      \ turn index bit off
        DUP  >R  C@  SWAP  2^  255 XOR  AND R>  C!  ;

: BIT-TOGGLE   ( index <name> --- )      \ toggle index bit
        2DUP  BIT? IF BIT-OFF ELSE BIT-ON  THEN ;
;S
(P BIT? yields the bit status. BIT-ON sets the bit. BIT-OFF
 sets the bit off. BIT-TOGGLE toggle the bit. Use the form:
     index  <name> BITxxxxxx     )

Victor-FORTH          By Timothy Huang         30 September 1982
```

Technotes (Continued)

Circular Lists

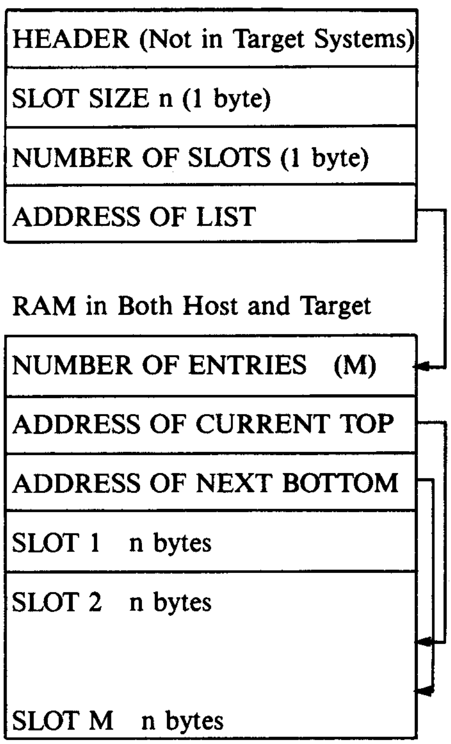
C.L. Stephens
COMSOL Ltd.

Editor's Note: The following technote was originally published in England as Computer Solution Ltd.'s "polyFORTH Note 19." As such it contains a few references to a multiprogrammed system. These references may be omitted.

Circular lists are a very convenient mechanism and once implemented can be used to generate stacks or FIFO buffers. The programs specified here are designed to set up and manipulate any number of circular lists. Each list may have up to 255 entries (or slots) and each slot may be up to 255 bytes long.

The word **CLIST** creates a named list with the following structure and **ILIST** initializes it.

RAM in Host or ROM in Target



The words **+TLIST** and **+BLIST** add entries at the top of the list, **-TLIST** and **-BLIST** remove entries.

In all cases the words return the address of the first byte in the slot allocated or to be released. The user must then move the required data into or from the slot.

Continued

```
120 LIST

( CIRCULAR LIST WORDS                                  CLS 30/ 9/81 )
CODE C+!  W POP  H POP  W LDAX  L ADD  W STAX  NEXT JMP
: CTOP  2+ @ 1+ ;                                  (current top)
: NXTBOT  2+ @ 3 + ;                               (next bottom)
: STOL  2+ @ 5 + ;                                (start of list)
: ENOL  DUP C@ OVER 1+ C@ 1- • SWAP STOL + ;       (end of list)
: +LIST  SWAP 2+ @ C+! ;               (changes the list count)
: ?SPACE  DUP 1+ C@  SWAP 2+ @ C@ - ;     (how many left in?)
: CLIST  CREATE  2DUP SWAP 256 * + ,         (creates a list)
     HERE 2+, (must be change to THERE, for target compile)
     • 5 + ALLOT  (reserves space)   DOES>   ;
91 LOAD   92 LOAD
```

```
121 LIST

( CIRCULAR LIST WORDS                                  CLS 17/ 9/81 )
: ILIST 0 OVER 2+ @ C! DUP STOL SWAP 2DUP NXTBOT !   CTOP! ;
                                         (initialise the list)
: WSPACE                       (wait for space in the list)
     BEGIN  DUP ?SPACE NOT  IF  PAUSE  AGAIN  DROP  ;
: WENTRY                    (wait for an entry in the list)
     BEGIN  DUP 2+ @ C@ NOT
     IF  PAUSE  AGAIN  DROP  ;
: +CIRCLE  OVER C@ +  2DUP SWAP ENOL >
     IF DROP DUP STOL  THEN  SWAP  ;
: -CIRCLE  OVER C@ -  2DUP SWAP STOL  <
     IF DROP DUP ENOL  THEN  SWAP  ;
```

```
122 LIST

( CIRCULAR LIST WORDS                                  CLS 18/ 9/81 )
: TLIST  DUP WSPACE
     DUP CTOP @ -CIRCLE
     2DUP CTOP !  1 +LIST ;
: -TLIST  DUP WENTRY
     DUP CTOP @ 2DUP +CIRCLE
     CTOP  ! SWAP -1 +LIST ;
: +BLIST  DUP WSPACE
     DUP NEXTBOT @ 2DUP +CIRCLE
     NXTBOT ! SWAP  1 +LIST ;
: -BLIST  DUP WENTRY
     DUP NXTBOT @ -CIRCLE
     2DUP NXTBOT ! -1 +LIST ;
```

```
123 LIST

( CIRCULAR LIST WORD   TESTS                           CLS 29/ 9/81 )
10 4 CLIST LOG
LOG ILIST
VARIABLE LOCAL 2 ALLOT
: DL  CR  LOG 1+ C@ DUP . 3 SPACES   LOG C@ DUP  .  3 SPACES
    LOG 2+ @ .    *    ( leaves number of entries for dump )
    CR LOG ?SPACE . 3 SPACES  LOG CTOP ? 3 SPACES LOG NXTBOT ?
    CR  LOG STOL SWAP  DUMP
: FILL 4 0 DO 2DUP I + C! LOOP 2DROP ;
: V+T LOG +TLIST FILL ;
: V+B LOG +BLIST FILL ;
: SHOW LOCAL 4 MOVE   ( moves the data to a local buffer )
    4 0 DO LOCAL I + C@ .  LOOP  ;
: V-T LOG -TLIST SHOW ;
: V-B LOG -BLIST SHOW ;
```

FORTH Classes

at
Humboldt State University
Eureka, California
by
Kim Harris

People, Computers, And FORTH Programming

5 Day Introductory Class

Date: June 20–24, 1983

Cost: $100 including 3 quarter hours of academic credit

Daily hands-on exercises

Outline:

I. The Forth development system
II. Developing applications

For people who want enough knowledge and experience with Forth to write application programs and understand some of the internal workings of a Forth system. Students should have some experience with one other computer language. Knowledge of assembly language is not required.

Using FORTH Effectively

5 Day Advanced Class

Date: June 27 - July 1, 1983

Cost: $150 including 3 quarter hours of academic credit

Daily hands-on exercises including a meta-compilation workshop.

Outline:

I. Forth system internals
II. Program development techniques

For people who want an in-depth understanding of the generation and internal operation of Forth systems. Students should have mastered an introductory Forth course or have 6 months experience with Forth, have mastered Starting Forth, and have knowledge of some assembly language and operating system principles.

Registration:

Send check or money order to:

Office of Continuing Education
Humboldt State University
Arcata, CA 95521

Include the course title(s) and the number of people registering. Enrollment will be filled on a first-received, first-served basis. Your payment will be returned if a course is full. You will receive a registration packet with additional information. Direct questions to the above office; their phone is (707) 826-3711. Ask for Claire Duffey.

Technotes (Continued)

These words assume a multi-task system and will wait if there is no space or no entry respectively. In single-task systems or emergency overrides **?SPACE** can be used to stop lockouts.

In a multi-task system it is important that a slot once allocated or released should be filled or emptied before the task executes a **PAUSE, MS** or I/O as other tasks might otherwise overwrite the areas.

Block 123 is a demonstration program that manipulates a list called **LOG** which has 10 entries with 4 bytes per entry.

DL displays the whole contents of this list. (May not be used multi tasking.) **FILL** and **SHOW** are used to fill the slots and display their contents respectively.

V+T and **V+B** take a value off the stack and put it into each of the bytes allocated to a slot requested at the top and bottom of the list.

V-T and **V-B** take a slot off the top or bottom of the list and prints its contents.

A typical application of these words is to have a terminal task driving a slow device such as a printer taking its data from the top of a circular list. Any tasks that wish to output messages to the printer may either add their lines to the bottom of the list or in a emergency may add them to the top of the list.

```
: RUN-LOG  TTY  ACTIVATE
    BEGIN CR LOG-FILE
    -TLIST  64 -TRAILING
    TYPE 0 END;
```

And in the other tasks this puts a local buffer into the list.

```
BUFFER  LOG-FILE
   +BLIST  64  CMOVE
```

RLOAD Program Package Load

C.L. Stephens
COMSOL
Chertsey, England

A problem frequently encountered when producing program packages is the need to move blocks to different areas of the disk in order to avoid conflicts with blocks already occupied. The FORTH word --> which loads the next block suffers from the disadvantage of consuming six bytes of return stack each time it is used. It also distributes the loading information across a range of blocks making changes difficult.

The preferred mechanism consists of an initial "Load Block" which acts as a directory to the application and as a central point for substitutions. However, this block, if it uses **LOAD**, will have to be extensively edited if moved to another system in which those blocks are already in use.

The word **RLOAD** takes a value from the stack, adds it to the number of the block in which the **RLOAD** is situated and then loads the resulting block

```
: RLOAD BLK @ + LOAD ;
```

This allows load blocks such as

```
1 RLOAD  2 RLOAD (Device drivers)
3 RLOAD  4 RLOAD (Control programs)
5 RLOAD          (Test programs)
```

An additional benefit of this word is that it allows position independent documentation. A description of the package can be in terms of the relative block numbers with a minor change to the program listing words allowing them to print relative rather than absolute block numbers.

This is a very useful technique. I've also seen the name **+LOAD**, *which I think came from Kim Harris. Although I speak out against abbreviations, I have a word in my own system called* **FH** *(for "from here") which performs* **BLK @ +** *but not* **LOAD**. *The syntax therefore is*

```
1 FH LOAD  2 FH LOAD
```

as in "one from here, load." I factored out **LOAD** *because there are other things you may want to do, like type text strings from disk. In any case, note that the argument may be positive or negative.*

Finally --> doesn't have to keep anything extra on the return stack because it doesn't have to re-invoke **LOAD**. *It merely must go*

```
ZERO >IN !  1 BLK +!
```

to jump the interpreter pointers to the top of the next block. —Leo Brodie

FIG Chapter News

Potomac Chapter

At the December 7th meeting we heard how Ted Beach, using just a baker's dozen of FORTH words, has created a simple-minded disk directory system that anyone can install in their FORTH environment. The word **DIR** will list out the entire directory of screens on a FORTH disk, while **FREE** will print out the number of unused screens available on the disk.

One of the most powerful features of the directory system is a redefinition of the FORTH word ' (tick). The new tick performs exactly as the standard tick, but will also search all available disk drive directories for the word before issuing an error message. If tick finds the word on any disk, it will load the associated screen(s) for you!

Ted demonstrated the system on the Radio Shack Color Computer.

At the January 4th meeting, Steven Knowles discussed "FORTH in Astronomy." Steve is an astronomer who for many years has used FORTH on minicomputers. Steve described his work with FORTH for data collecting and shared his recollections and ideas about FORTH.

Las Vegas Chapter

The third formal meeting of the Las Vegas FIG was held Monday, 8 November 1982 at 7 P.M., Valley Bank Center, 101 Convention Center Drive, Suite 900.

The main subject of this meeting was coding forms. There were free handouts of introductory FORTH literature, and many FORTH tutorial books and articles were available for browsing. Also featured was a demonstration of MVP-FORTH on the IBM Personal Computer.

Australia Chapter

Lance Collins, Secretary

The Australian Fig Chapter meets in Melbourne at the home of the secretary on the first Friday of each month. We have been going since mid-1981 and have a core membership of about 10. We have welcomed about 30 visitors in 1982. We find it difficult to hold new members as there is little serious FORTH activity here, and we are sometimes embarassed by the question, "But what real applications are you running in FORTH?" Fortunately, this is changing and 1983 looks most promising for FORTH in Australia.

The availability of FORTH books and software is limited here and the Chapter has established a comprehensive library of FORTH books for the use of our members. We also make available FORTH software, and the secretary is an agent for Mountain View Press. We have a catalog we send to enquirers which describes mainly commercial books and software, as we have not yet found the resources to classify and package the mass of public domain FORTH items we are accumulating.

By the time this is published there may be a Sydney group under way. Contact Peter Tregeagle (02)524 7490 for details.

Start Your Own FIG Chapter

What is a FIG Chapter?

There are two kinds of FIG chapters: local, and special-interest. Local chapters are centered in a city or region. Special-interest chapters may be non-geographical; they focus on an interest area such as an application (e.g., robotics, telecommunication), or on FORTH for a particular computer.

All chapters must provide a contact point, and some form of regular public access (usually meetings). Non-geographical chapters will normally provide other forms of access, such as a newsletter or telecommunications, instead of meetings.

Why Have a FIG Chapter?

A chapter lets you share information with other FORTH users in your geographical or application area. In addition, FIG provides several specific benefits:

(A) FIG will list your chapter in *FORTH Dimensions*, so that others can find your group.

(B) *FORTH Dimensions* will give priority to publishing chapter news, which can help you make professional contacts in the areas of your particular interests.

(C) FIG will occasionally supply material, such as meeting handouts or tapes, which can serve as a discussion topic at local meetings.

(D) FIG will supply its publications at bulk rates; local chapters can sell them to raise money, and to provide immediate local access to the material.

(E) Chapters can apply to FIG for one-time funding for activities.

How to Start a FIG Chapter

To be recognized as a chapter, a group must have (1) a contact person, (2) regular public access (usually by meetings which are open to the public), and (3) at least five members of FIG. If you don't know five members in your area, FIG can help you contact them. If you want to start a chapter, send a request for a FIG Chapter Kit to the Chapter Coordinator, FORTH Interest Group, P.O. Box 1105, San Carlos, CA 94070.

FIG Chapters

U.S.

• ARIZONA

Phoenix Chapter
Dennis L. Wilson
Samaritan Health Services
2121 E. Magnolia
Phoenix, AZ
602/257-6875

• CALIFORNIA

Los Angeles Chapter
Monthly, 4th Sat., 11 a.m.
Allstate Savings
8800 So. Sepulveda Boulevard
Los Angeles
Phillip Wasson
213/649-1428

Northern California Chapter
Monthly, 4th Sat., 1 p.m.
FORML Workshop at 10 a.m.
Palo Alto area.
Contact FIG Hotline
415/962-8653

Orange County Chapter
Monthly, 4th Wed., 12 noon.
Fullerton Savings
18020 Brookhurst
Fountain Valley
714/523-4202

San Diego Chapter
Weekly, Thurs., 12 noon.
Call Guy Kelly
714/268-3100 ext.4784

• MASSACHUSETTS

Boston Chapter
Monthly, 1st Wed., 7 p.m.
Mitre Corp. Cafeteria
Bedford, MA
Bob Demrow
617/688-5661 after 5 p.m.

• MICHIGAN

Detroit Chapter
Call Dean Vieau
313/493-5105

• MINNESOTA

MNFIG Chapter
Monthly, 1st Mon.
MNFIG
1156 Lincoln Avenue
St. Paul, MN 55105
Call Mark Abbot (days)
612/854-8776 or
Fred Olson
612/588-9532

• MISSOURI

St. Louis Chapter
Call David Doudna
314/867-4482

• NEVADA

Las Vegas Chapter
Suite 900
101 Convention Center Drive
Las Vegas, NV 89109
702/737-5670

• NEW JERSEY

New Jersey Chapter
Call George Lyons
201/451-2905 eves.

• NEW YORK

New York Chapter
Call Tom Jung
212/746-4602

• OKLAHOMA

Tulsa Chapter
Monthly, 3rd Tues., 7:30 p.m.
The Computer Store
4343 South Peoria
Tulsa, OK
Call Bob Giles
918/599-9304 or
Art Gorski
918/743-0113

• OHIO

Dayton Chapter
Monthly, 2nd Tues.
Datalink Computer Center
4920 Airway Road
Dayton, OH 45431
Call Gary Granger
513/849-1483

• OREGON

Portland Chapter
Call Timothy Huang
9529 Northeast Gertz Circle
Portland, OR 97211
503/289-9135

• PENNSYLVANIA

Philadelphia Chapter
Continental Data Systems
1 Bala Plaza, Suite 212
Bala Cynwid, PA 91004
Call Barry Greebel

• TEXAS

Austin Chapter
Call John Hastings
512/327-5864

Dallas/Ft. Worth Chapter
Monthly, 4th Thurs., 7 p.m.
Software Automation
1005 Business Parkway
Richardson, TX
Call Marvin Elder
214/231-9142 or
Bill Drissel
214/264-9680

• UTAH

Salt Lake City Chapter
Call Bill Haygood
801/942-8000

• VERMONT

Vermont Fig Chapter
Monthly, 3rd Mon., 7:30 p.m.
Vergennes Union High School
Room 210, Monkton Road
Vergennes, VT 05491
Contact Hal Clark
RD #1 Box 810
Starksboro, VT 05487
802/877-2911 days;
802/453-4442 eves.

• VIRGINIA

Potomac Chapter
Monthly, 1st Tues., 7 p.m.
Lee Center
Lee Highway at Lexington St.
Arlington, VA
Call Joel Shprentz
703/437-9218 eves.

FOREIGN

• AUSTRALIA

Australia Chapter
Contact Lance Collins
65 Martin Road
Glen Iris, Victoria 3146
(03)292600

• CANADA

Southern Ontario Chapter
Contact Dr. N. Solntseff
Unit for Computer Science
McMaster University
Hamilton, Ontario L8S 4K1
416/525-9140 ext. 2065

Quebec Chapter
Call Gilles Paillard
418/871-1960 or
418/643-2561

• ENGLAND

English Chapter
FORTH Interest Group
38 Worsley Road
Frimley, Camberley
Surrey, GU16 5AU, England

• JAPAN

Japanese Chapter
Masa Tasaki
Baba-Building 8F
3-23-8 Nishi-Shimbashi
Minato-ku, Tokyo
105 Japan

• NETHERLANDS

HCC-FORTH Interest Group Chapter
F.J. Meijer
Digicos
Aart V.D. Neerweg 31
Ouderkerk A.D.
Amstel, The Netherlands

• WEST GERMANY

West German Chapter
Klaus Schleisiek
FIG Deutschland
Postfach 202264
D 2000 Hamburg 20
West Germany

SPECIAL GROUPS

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

Detroit Atari FORTH
Monthly, 1st Wed.
Call Tom Chrapkiewicz
313/524-2100 or
313/772-8291

Nova Group Chapter
Contact Mr. Francis Saint
2218 Lulu
Witchita, KS 67211
316/261-6280 days

MMSFORTH Users Chapter
Monthly, 3rd Wed., 7 p.m.
Cochituate, MA
Dick Miller
617/653-6136

FORTH System Vendors (by Category)

(Codes refer to alphabetical listing e.g., A1 signifies AB Computers, etc.)

Processors

1802	C1, C2, F3, F6, L3
6502 (AIM, KIM, SYM)	R1, R2, S1
6800	C2, F3, F5, K1, L3, M6, T1
6801	P4
6809	C2, F3, L3, M6, S11, T1
68000	C2, C4, D1, E1, K1
68008	P4
8080/85	A5, C1, C2, F4, I5, L1, L3, M3, M6, R1, T3
Z80/89	A3, A5, C2, F4, I3, L1, M2, M3, M5, N1, T3
Z80000	I3
8086/88	C2, F2, F3, L1, L3, M6
9900	E2, L3

Operating Systems

CP/M	A3, A5, C2, F3, I3, L3, M1, M2, M6, T3
CP/M86	C2

Computers

Alpha Micro	P3, S3
Apple	A4, F4, I2, I4, J1, L4, M2, M6, 02, 03
Atari	M6, P2, Q1, V1
Cromemco	A5, M2, M6
DEC PDP/LSI-11	C2, F3, L2, S3
Heath-89	M2, M6
Hewlett-Packard 85	
Hewlett-Packard 9826/36	C4
IBM PC	A8, C2, F3, L1, M5, M6, Q2, S9, W2
IBM Other	L3, W1
Kaypro II/Xerox 820	M2
Micropolis	A2, M2, S2
North Star	I5, M2, P1, S7
Nova	C5
Ohio Scientific	A6, B1, C3, O1, S6, T2
Osborne	M2
Pet SWTPC	A1, A6, B1, C3, O1, S6, T2, T5
Poly Morphic Systems	A7
TRS-80 I, II, III	I5, M2, M5, M6, S4, S5, S10
TRS-80 Color	A3, A8, F5, M4, S11, T1
Vector Graphics	M2

Other Products/Services

Applications	P4
Boards, Machine	F3, M3, P4, R2
Consultation	C2, C4, N1, P4, T3, W1
Cross Compilers	C2, F3, I3, M6, N1, P4
Products, Various	A5, C2, F3, I5, S8, W2
Training	C2, F3, I3, P4, W1

FORTH System Vendors (Alphabetical)

The following vendors offer FORTH systems, applications, or consultation. FIG makes no judgement on any product, and takes no responsibility for the accuracy of this list. We encourage readers to keep us informed on availability of the products and services listed. Vendors may send additions and corrections to the Editor, and must include a copy of sales literature or advertising.

FORTH Systems

1. AB Computers
   252 Bethlehem Pike
   Colmar, PA 18915
   215/822-7727
2. Acropolis
   17453 Via Valencia
   San Lorenzo, CA 94580
   415/276-6050
4. Applied Analytics Inc.
   8910 Brookridge Dr., #300
   Upper Marlboro, MD 20870
5. Aristotelian Logicians
   2631 E. Pinchot Ave.
   Phoenix, AZ 85016
7. Abstract Systems, etc.
   RFD Lower Prospect Hill
   Chester, MA 01011
8. Armadillo Int'l Software
   P.O. Box 7661
   Austin, TX 78712
   512/459-7325

B

1. Blue Sky Products
   729 E. Willow
   Signal Hill, CA 90806

C

1. CMOSOFT
   P.O. Box 44037
   Sylmar, CA 91342
2. COMSOL, Ltd.
   Treway House
   Hanworth Lane
   Chertsey, Surrey
   England KT16 9LA
3. Consumer Computers
   8907 La Mesa Blvd.
   La Mesa, CA 92041
   714/698-8088
4. Creative Solutions, Inc.
   4801 Randolph Rd.
   Rockville, MD 20852
5. Capstone Computing, Inc.
   5640 Southwyck Blvd., #2E
   Toledo, OH 43614
   419/866-5503

E

1. Emperical Research Group
   P.O. Box 1176
   Milton, WA 98354
   206/631-4855
2. Engineering Logic
   1252 13th Ave.
   Sacramento, CA 95822

F

1. Fantasia Systems, Inc.
   1059 The Alameda
   Belmont, CA 94002
   415/593-5700
3. FORTH, Inc.
   2309 Pacific Coast Highway
   Hermosa Beach, CA 90254
   213/372-8493
4. FORTHWare
   639 Crossridge Terrace
   Orinda, CA 94563
5. Frank Hogg Laboratory
   130 Midtown Plaza
   Syracuse, NY 13210
   315/474-7856
6. FSS
   P.O. Box 8403
   Austin, TX 78712
   512/477-2207

I

1. IDPC Company
   P.O. Box 11594
   Philadelphia, PA 19116
   215/676-3235
2. IUS (Cap'n Software)
   281 Arlington Ave.
   Berkeley, CA 94704
   415/525-9452
3. Inner Access
   517K Marine View
   Belmont, CA 94002
   415/591-8295
4. Insoft
   10175 S.W. Barbur Blvd.
   Suite #202B
   Portland, OR 97219
   503/244-4181
5. Interactive Computer
   Systems, Inc.
   6403 Di Marco Rd.
   Tampa, FL 33614

J

1. JPS Microsystems, Inc.
   361 Steelcase Rd., W.
   Markham, Ontario
   Canada L3R 3V8
   416/475-2383

K

1. Kukulies, Christoph
   Ing. Buro Datentec
   Heinrichsallee 35
   Aachen, 5100
   West Germany

FORTH System Vendors

L

1. Laboratory Microsystems
   4147 Beethoven St.
   Los Angeles, CA 90066
   213/306-7412
2. Laboratory Software Systems, Inc.
   3634 Mandeville Canyon
   Los Angeles, CA 90049
   213/472-6995
3. Lynx
   3301 Ocean Park, #301
   Santa Monica, CA 90405
   213/450-2466
4. Lyons, George
   280 Henderson St.
   Jersey City, NJ 07302
   201/451-2905

M

1. M & B Design
   820 Sweetbay Dr.
   Sunnyvale, CA 94086
2. MicroMotion
   12077 Wilshire Blvd., #506
   Los Angeles, CA 90025
   213/821-4340
3. Microsystems, Inc.
   2500 E. Foothill Blvd., #102
   Pasadena, CA 91107
   213/577-1477
4. Micro Works, The
   P.O. Box 1110
   Del Mar, CA 92014
   714/942-2400
5. Miller Microcomputer
   61 Lake Shore Rd.
   Natick, MA 01760
   617/653-6136
6. Mountain View Press
   P.O. Box 4656
   Mountain View, CA 94040
   415/961-4103

N

1. Nautilus Systems
   P.O. Box 1098
   Santa Cruz, CA 95061
   408/475-7461

O

1. OSI Software & Hardware
   3336 Avondale Court
   Windsor, Ontario
   Canada N9E 1X6
   519/969-2500
2. Offete Enterprises
   1306 S "B" St.
   San Mateo, CA 94402
3. On-Going Ideas
   RD #1, Box 810
   Starksboro, VT 05487
   802/453-4442

P

1. Perkel Software Systems
   1636 N. Sherman
   Springfield, MO 65803
2. Pink Noise Studios
   P.O. Box 785
   Crockett, CA 94525
   415/787-1534
3. Professional Mgmt. Services
   724 Arastradero Rd., #109
   Palo Alto, CA 94306
   408/252-2218
4. Peopleware Systems Inc.
   5190 West 76th St.
   Minneapolis, MN 55435
   612/831-0872

Q

1. Quality Software
   6660 Reseda Blvd., #105
   Reseda, CA 91335
2. Quest Research, Inc.
   P.O. Box 2553
   Huntsville, AL 35804
   800/558-8088

R

2. Rockwell International
   Microelectronics Devices
   P.O. Box 3669
   Anaheim, CA 92803
   714/632-2862

S

1. Saturn Software, Ltd.
   P.O. Box 397
   New Westminister, BC
   V3L 4Y7 Canada
2. Shaw Labs, Ltd.
   P.O. Box 3471
   Hayward, CA 94540
   415/276-6050
3. Sierra Computer Co.
   617 Mark NE
   Albuquerque, NM 87123
4. Sirius Systems
   7528 Oak Ridge Highway
   Knoxville, TN 37921
   615/693-6583
5. Software Farm, The
   P.O. Box 2304
   Reston, VA 22090
6. Software Federation
   44 University Drive
   Arlington Hts., IL 60004
   312/259-1355
7. Software Works, The
   1032 Elwell Ct., #210
   Palo Alto, CA 94303
   415/960-1800
8. Supersoft Associates
   P.O. Box 1628
   Champaign, IL 61820
   217/359-2112
9. Satellite Software Systems
   288 West Center
   Orem, UT 84057
   801/224-8554
10. Spectrum Data Systems
    5667 Phelps Luck Dr.
    Columbia, MD 21045
    301/992-5635
11. Stearns, Hoyt Electronics
    4131 E. Cannon Dr.
    Phoenix, AZ 85028
    602/996-1717

T

1. Talbot Microsystems
   1927 Curtis Ave.
   Redondo Beach, CA 90278
2. Technical Products Co.
   P.O. Box 12983
   Gainsville, FL 32604
   904/372-8439
3. Timin Engineering Co.
   6044 Erlanger St.
   San Diego, CA 92122
   714/455-9008
4. Transportable Software
   P.O. Box 1049
   Hightstown, NJ 08520
   609/448-4175

V

1. Valpar International
   3801 E. 34th St.
   Tucson, AZ 85713
   800/528-7070

W

1. Ward Systems Group
   8013 Meadowview Dr.
   Frederick, MD 21701
2. Worldwide Software
   2555 Buena Vista Ave.
   Berkeley, CA 94708
   415/644-2850

Z

1. Zimmer, Tom
   292 Falcato Dr.
   Milpitas, CA 95035

Boards & Machines Only

See System Vendor Chart
for others

Controlex Corp.
16005 Sherman Way
Van Nuys, CA 91406
213/780-8877

Datricon
7911 NE 33rd Dr., #200
Portland, OR 97211
503/284-8277

Golden River Corp.
7315 Reddfield Ct.
Falls Church, CA 22043

Application Packages Only

See System Vendor Chart
for others

Curry Associates
P.O. Box 11324
Palo Alto, CA 94306
415/322-1463

InnoSys
2150 Shattuck Ave.
Berkeley, CA 94704
415/843-8114

Consultation & Training Only

See System Vendor Chart
for others

Boulton, Dave
581 Oakridge Dr.
Redwood City, CA 94062

Brodie, Leo
9720 Baden Ave.
Chatsworth, CA 91311
213/998-8302

Eastgate Systems Inc.
P.O. Box 1307
Cambridge, MA 02238

Girton, George
1753 Franklin
Santa Monica, CA 90404
213/829-1074

Go FORTH
504 Lakemead Way
Redwood City, CA 94062
415/366-6124

Harris, Kim R.
Forthright Enterprises
P.O. Box 50911
Palo Alto, CA 94303
415/858-0933

Laxen, Henry H.
1259 Cornell Ave.
Berkeley, CA 94706
415/525-8582

McIntosh, Norman
2908 California Ave., #3
San Francisco, CA 94115
415/563-1246

Metalogic Corp.
4325 Miraleste Dr.
Rancho Palos Verdes, CA 90274
213/519-7013

Petri, Martin B.
15508 Lull St.
Van Nuys, CA 91406
213/908-0160

Redding Co.
P.O. Box 498
Georgetown, CT 06829
203/938-9381

Schleisiek, Klaus
Eppendorfer Landstr. 16
D 2000 Hamburg 20
West Germany
(040)480 8154

Schrenk, Dr. Walter
Postfach 904
7500 Karlstruhe-41
West Germany

Software Engineering
317 W. 39th Terrace
Kansas City, MO 64111
816/531-5950

Technology Management, Inc.
1520 S. Lyon St.
Santa Ana, CA 92705
714/835-9512

FORTH INTEREST GROUP

MAIL ORDER

	USA	FOREIGN AIR
☐ Membership in FORTH Interest Group and Volume V of FORTH DIMENSIONS	$15	$27
☐ Back Volumes of FORTH DIMENSIONS. **Price per each.** ☐ I ☐ II ☐ III ☐ IV	$15	$18
☐ fig-FORTH Installation Manual, containing the language model of fig-FORTH, a complete glossary, memory map and installation instructions	$15	$18
☐ Assembly Language Source Listings of fig-FORTH for specific CPU's and machines. The above manual is required for installation. Check appropriate box(es). **Price per each.** ☐ 1802 ☐ 6502 ☐ 6800 ☐ 6809 ☐ VAX ☐ Z80 ☐ 8080 ☐ 8086/8088 ☐ 9900 ☐ APPLE II ☐ ECLIPSE ☐ PACE ☐ NOVA ☐ PDP-11 ☐ 68000 ☐ ALPHA MICRO	$15	$18
☐ "Starting FORTH" by Brodie. BEST book on FORTH. (Paperback)	$18	$22
☐ "Starting FORTH" by Brodie. (Hard Cover)	$22	$27
☐ PROCEEDINGS 1980 FORML (FORTH Modification Lab) Conference	$25	$35
☐ PROCEEDINGS 1981 FORML Conference, Both Volumes	$40	$55
☐ Volume I, Language Structure	$25	$35
☐ Volume II, Systems and Applications	$25	$35
☐ PROCEEDINGS 1982 FORML Conference	$25	$35
☐ PROCEEDINGS 1981 FORTH Univ. of Rochester Conference	$25	$35
☐ PROCEEDINGS 1982 FORTH Univ. of Rochester Conference	$25	$35
☐ FORTH-79 Standard, a publication of the FORTH Standards Team	$15	$18
☐ Kitt Peak Primer, by Stevens. An in-depth self-study primer.	$25	$35
☐ BYTE Magazine Reprints of FORTH articles, 8/80 to 4/81	$ 5	$10
☐ FIG T-shirts: ☐ Small ☐ Medium ☐ Large ☐ X-Large	$10	$12
☐ Poster, August 1980 BYTE cover, 16" x 22"	$ 3	$ 5
☐ FORTH Programmer's Reference Card. If ordered separately, send a stamped, addressed envelope.	FREE	
☐ Dr. Dobb's Journal, Two FORTH Issues, 9/81 & 9/82	$ 7	$10
TOTAL	$________	

NAME______________________________ MAIL STOP/APT__________

ORGANIZATION______________________ PHONE (   )__________

ADDRESS____

CITY__________________ STATE________ ZIP________ COUNTRY________

VISA #__________________ MASTERCARD #________________________

Expiration Date_______________ (Minimum of $15.00 on charge cards)

Make check or money order in US Funds on US bank, payable to: FIG. All prices include postage. **No purchase orders without check.** California residents add sales tax. 1/83

ORDER PHONE NUMBER: (415) 962-8653

FORTH INTEREST GROUP * PO BOX 1105 * SAN CARLOS, CA 94070

FORTH INTEREST GROUP

P.O. Box 1105
San Carlos, CA 94070

BULK RATE
U.S. POSTAGE
PAID
Permit No. 261
Mt. View, CA

PLEASE ADVISE CORRESPONDENT
OF YOUR CURRENT MAILSTATION #

RENEW YOUR SUBSCRIPTION !