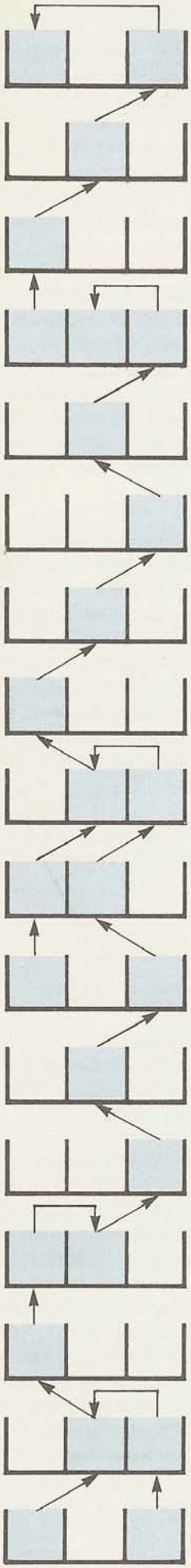

FORTH DIMENSIONS

FORTH INTEREST GROUP
P.O. Box 1105
San Carlos, CA 94070

Volume III
Number 3
Price $2.50

INSIDE

FORTH DIMENSIONS

Published by Forth Interest Group

Volume III No. 3 September/October 1981

Publisher Roy C. Martens
Editor C. J. Street

FORTH DIMENSIONS solicits editorial material, comments and letters. No responsibility is assumed for accuracy of material submitted. ALL MATERIAL PUBLISHED BY THE FORTH INTEREST GROUP IS IN THE PUBLIC DOMAIN. Information in FORTH DIMENSIONS 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

HISTORICAL PERSPECTIVE

FORTH was created by Mr. Charles H. Moore in 1969 at the National Radio Astronomy Observatory, Charlottesville, VA. It was created out of dissatisfaction with available programming tools, especially for observatory automation.

Mr. Moore and several associates formed FORTH, Inc. in 1973 for the purpose of licensing and support of the FORTH Operating System and Programming Language, and to supply application programming to meet customers' unique requirements.

The Forth Interest Group is centered in Northern California. Our membership is over 2,400 worldwide. It was formed in 1978 by FORTH programmers to encourage use of the language by the interchange of ideas through seminars and publications.

EDITOR'S COLUMN

This month introduces the long-promised MARKETING COLUMN. Considering that one of the best ways to proliferate FORTH is to sell it in the form of applications, I believe this column will contribute to the financial welfare of our members and help make the FORTH language a force in the software world. Questions related to all phases of marketing from product research and development to pricing, advertising and distribution channels are welcome.

Our next edition will be devoted to the conference at the University of Rochester and will be coordinated by Mr. Larry Forsely of that institution. One of my main goals as editor has been to "de-Californize" FORTH DIMENSIONS and make it reflect more accurately the opinions and techniques employed throughout the FORTH world. This next edition is a bold experiment in this regard and I have every confidence in Mr. Forsely helping to make it a successful one.

The issue following ROCHESTER will be devoted to music, graphics and games. Currently, this type of application is in very short supply and I am appealing to our members to submit them as soon as possible. Please remember, YOU DON'T HAVE TO BE A WRITER—our staff will help provide whatever is necessary to make your article or game publishable—but the raw ideas and code have to come from you. Also, we are not necessarily looking for lengthy, complex and elegant applications—simple, instructive, short codes often are best and the most useful.

Please contribute whatever you can—ultimately the quality and utility of FORTH DIMENSIONS comes from our members!

C. J. Street
Editor

PUBLISHER'S COLUMN

Nov. 2-4: Mini/Micro Show, Anaheim, CA
Nov. 25-27: FORML Conference, Pacific Grove, CA
Nov. 28: FIG National Convention, Santa Clara, CA
Mar. 19-21: Computer Faire, San Francisco, CA

FORTH vendors—these shows can be helpful to you in several ways. First, if you will send FIG approximately 500 flyers, 8½ x 11, about your products, we'll display them at all four places. Second, you should exhibit at the FIG Convention on November 28 at the Marriott Hotel, Santa Clara, CA. An 8' table is only $50.00—send a check to FIG, today. Third, FIG has a prime location at Computer Faire, March 19-21, 1982 in San Francisco. We have booths 1343C and 1442C; these face the central booth area and form an island with eight other booths. Six of these booths are currently available. Lets get all FORTH vendors together. All you have to do is call Computer Faire (415) 851-7075 and tell them you want to be in the same island as the FORTH Interest Group.

Roy Martens

LETTERS

Dear Fig:

I am disturbed to see that you have recently published a review comparing our Z-80 FORTH package to TIMINS FORTH and stating that the Z-80 FORTH is significantly slower than the 8080 version. Please be informed that the benchmark was run on an early version of Z-80 FORTH which has not been distributed for the last nine months. The current version that we are selling has been benchmarked by some of our customers as 5-10% FASTER than 8080 FORTH.

We have made an effort to provide a high quality, comprehensive FORTH program development package at a very reasonable cost. Over 300 copies of the current Z-80 FORTH version have been sold so far and I do not know of any unsatisfied customers. I think it reflects very poorly on your publication that you would print such a review without checking the facts with all of the interested parties. If you people are really concerned with promoting the FORTH language, please be a little more careful with attacking the reputation and products of the FORTH language vendors.

Ray Duncan
LABORATORY MICROSYSTEMS
4147 Beethoven St.
Los Angeles, CA 90066

We fail to see how publishing a comparison review of products that were on the market less than a year ago can be construed as an attack on either that product or the reputation of its vendor.

No inference has been made that Laboratory Microsystems Z-80 FORTH is an inferior product, or that it has dissatified customers. In fact, if the mail we are receiving from your customers is any indication, quite the contrary is true. Admittedly, the review published failed to indicate which versions were being compared, but we know of no facts that were inaccurately reported. If we are to provide the FORTH world with useful product reviews, accuracy is important, and if Laboratory Microsystems knows of any inaccurate facts or would care to provide us with an updated review, ample space will be provided.--ed.

Dear Fig:

Let me introduce myself: I'm Jim Gerow, an avid MMS FORTH user, a FORTH programmer (installer) for larger machines, and a member of the local MMS FORTH user group.

I've been referred to you by Dick Miller of MMS as a correspondent. Please let me know how I can be of service and how our MMS FORTH User Group can support you.

Jim Gerow
1630 Worcester Rd., #630C
Framingham, MA 01701

Thank you--we would appreciate any articles, ideas, bug fixes, or usable programs or tools you can send in. Looking forward to hearing from you--ed.

Dear Fig:

San Diego has a FORTH Interest Group that meets informally each Thursday and somewhat formally the 4th Tuesday of each month.

Because of the different machines, MPU's, and operating systems, (i.e., Disk or Cassette, etc.) we have a problem of software exchange (transportability). We are considering the development of a communications package involving RS-232 modems and software.

We are interested in finding out what FIG has, if anything, in standardizing any of the communications such as protocol or hardware specification.

Currently, most of the software exchange involves hardcopy. If you have any information relative to communications between FORTH operating machines or protocol standards used, we would appreciate your help.

K. G. Busch
Rancho Bernardo
12615 Higa Place
San Diego, CA 92128

O.K. members--how about giving Ken a hand? Or better yet, send the info to FORTH DIMENSIONS and we will publish it for all of our members' use.--ed.

Dear Fig:

Would you please forward me a writer's kit? I'm thinking about writing something for Nov./Dec. GRAPHIC/MUSIC. I have implemented a set of graphic words for Columbia Data Products' MX-964 (Z-80 Micro-*2, 512x256 bit mapped, $10-8080-figFORTH), and am working on some musical words for a dual GI's sound chip board. If I can get my hands on Digitalk (National Semi) early enough, maybe some work also can be done on that before the date I send out my articles.

Since you share the very same address as the 8080 Renovation Project, would you please forward the following page to them for me? Many thanks. Happy FORTH!

Tim Huang
9529 N.E. Gertz Circle
Portland, OR 97211

Thanks, Tim--we'll be in touch. The graphics issue is approaching rapidly (deadline is Oct. 15, 1981)--word to the wise--ed.

Dear Fig:

This is a note authorizing you to use the cartoon-style illustrations in the book Starting FORTH by Leo Brodie of FORTH, Inc. The credit line should read Leo Brodie, FORTH, Inc., Starting FORTH, a forthcoming Prentice-Hall publication. Reprinting by permission of Prentice-Hall, Inc., Englewood Cliffs, N.J. After October 1, 1981 you can leave the "forthcoming" out since the book will be in print. Thank you for your interest.

Jim F. Fegen, Jr.
Editor, The Computing Sciences
PRENTICE-HALL
Englewood Cliffs, NJ 07632

Thank you Prentice-Hall. Watch for cartoons from this important work.--ed.

Dear Fig:

Here is your complimentary copy of Starting FORTH. We at FORTH, Inc. hope you enjoy it as much as Mark Garetz of INFOWORLD, who said it was the best beginner's book he'd seen.

The hard- and soft-bound editions by Prentice-Hall will be on the shelves Sept. 8.

Let us know what you think of the book. We are anxious to hear your comments.

Winnie Shows
Public Relations
FORTH, Inc.
2309 Pacific Coast Highway
Hermosa Beach, CA 90254

Thanks, Winnie. Please note the review in this issue.--ed.

Dear Fig:

I live in a country town in Australia and the number of local computer hobbyists can be counted on one hand. I have so far converted one friend to FORTH and we have found all the back issues of FORTH DIMENSIONS very helpful with programming examples. I have had my system for about 5 years; it is an S100 Z-80 system with recently added dual Micropolis Mod II disks. I have rewritten the 8080 FIG FORTH CP/M interface to work with Micropolis DOS and am currently reworking some of the 8080 CODE definitions to use Z-80 instructions where they will improve the code. I am interested in corresponding with other FIG users, particularly those with systems similar to mine.

I wish to make a comment about the naming of words related to 32-bit integer operations. The present mixture of prefixed "D" and "2" make these words more frustrating to learn and use. That would not be the case with consistent prefix character. I think that the prefix character should be "D" for double. I am sure that most of us find the prefix letter "C" easy to use for 8 bit operations and I am glad the ASCII did not allow 1/2@ to be used. When floating point comes around (for example, in a 6 byte format), it seems most likely that F! will be used, not 3!. So let's be consistent and leave digits for numbers and use a prefix letter mnemonic to indicate stack operations, etc. that are not the usual one word (16 bits).

Could someone please explain what the HEX value A081 is for, in the definition of VOCABULARY? I can't work it out.

Keep up the good work with the magazine.

Bill Miles
PO Box 225
Red Cliffs
Victoria 3496
Australia

Thanks for your comments, Bill. Glad to hear FORTH is alive and well in the land down under! How about some of our members corresponding with Bill and helping him over the rough spots.--ed.

Dear Fig:

FORTH DIMENSIONS has grown increasingly useful to me in the past few months as I have finally begun to "get the hang of" FORTH. I have running on my TRS-80 several versions of BASIC, FORTRAN, PASCAL, APL, SAM76 and LISP under both TRSDOS and CDOS; but I have never found any language harder to learn that FORTH. Part of the problem is the scope of FORTH: at the same time I'm trying to understand the interpreter, compiler, OS, and a syntax as difficult as LISP's. I have found all the instruction manuals so far to make a drastic jump from simple concepts like 2 2 + . and DO...LOOPS to discussions of the Dictionary and of Defining Words. (I think the writers had the same problems I have, of separating the various functions of the system.) One of the best helps I received was Mr. Bumgarner's Stack Diagram in this year's March issue of BYTE. The necessity of being able to visualize the stack cannot be overemphasized. Once I was able to do that, I starting learning in earnest.

Having leaped this hurdle, I found FORTH more rewarding than any other language to learn. One of its greatest advantages to me as an applications programmer is its (almost) utterly consistent syntax: operators, functions, and "procedures" disappear and all you have are Words that get their arguments off the stack and place their return values on the stack. All my applications so far have been in BASIC. They are as "structured" as I can make them (i.e., subroutines calling subroutines), but it soon becomes hard to remember what "GOSUB 12000" does and which variables have to have what values in order to do whatever to whom and where! Not so with FORTH: although a restricted use of variables does make the program somewhat less readable, keeping most of the arguments and returned values on the stack actually makes it more "writeable" because there is no need to remember what the formal or actual parameters are. Right now, because of my limited experience with FORTH, it takes longer to write a "routine" than it would in BASIC: but already the total time to test and debug is far less.

I'm using Miller Microcomputer Services' top-notch MMSFORTH, and I have absolutely nothing bad to say about these people. Last summer I drove down to New England in order to pick up some hardware, and decided to drop in on Richard and Jill Miller in Natick, MA. They showed me the utmost in hospitality, helping me purchase equipment and wasting their time in general to make sure that my trip was worthwhile. Their product is excellent: worth it at twice the price and more (you didn't hear that, Dick!)--with standard features such as Strings, Double-Precision, Graphics, a good Screen Editor, and not one, but several fine demonstrations programs. A++ for MMSFORTH.

Morningstar is a software house in southern Ontario that does mainly custom programming. All of it so far has been BASIC, but we expect to have fully switched to FORTH by the end of 12 months, D.V. No other language would have compelled us to give up "Tandy Compatibility," but the advantages of FORTH far outweigh any extra cost for the lanage.

Thanks for your attention.

Vincent Otten
MORNINGSTAR
225 Dundas St.
Woodstock, Ontario
CANADA N4S 1A8

I am sure Dick Miller appreciates your comments. You might also look into Mr. Leo Brodie's new book Starting FORTH (reviewed elsewhere in this issue) that will be available in mid-September.--ed

Dear Fig:

I very much enjoyed my first pass through your article "Compiler Security" in FORTH DIMENSIONS III/1. I plan to re-read the article when I have more time.

In terms of the multi-user environment, haven't you almost answered your own question of security always versus security on demand with your parenthetical "and the other users" remark? This was near the end, in the discussion of the possible use of a "Novice Programmer Protection" package. In a single-user environment, more liberties can be taken, but I know I'm a novice user, having only been involved with computers since 1958 or so, and having only "FORTHed" non-intensively for about 3 years. My single-user system would always include the protection package (well. . .almost always). I would not, however, object to making security optional in the single-user case (but I am not a prospect for a FORTH implementation without it).

I don't agree with your characterization of Assembler security as inappropriate. It is the ability to have unstructured code that causes many of the problems with assembly code. If it is so easy as to be tempting, we will all err. FORTH makes the cost-to-fix versus time-error-found curve perhaps less steep, but early error detection is still cheaper, and software is still the largest part of the system cost (and getting larger). I cannot argue against being able to defeat Assembler security fairly easily, however, since there may be situations in which the risk is worth it.

John W. Baxter
Sr. Principal Programmer Analyst
NCR CORPORATION
Coronado, CA

MR. SHAW REPLIES:

I hope that after three years of FORTH programming that you have developed good FORTH style. This should be the case unless you have let your previous 20 years of experience interfere with your learning of FORTH's simplistic concepts.

In either case, you should be aware that good FORTH code is well thought out and very short. Most definitions, in either high level or assembler, should be very short; not more than a few lines. In very few instances is high level code ever longer. Those definitions that are long should be so well scrutinized as to the reasons for their length that the type of errors that the current compiler security would trap should not exist. Assembler code should only be used when speed is a critical factor. And then, structured code may not be the easiest or fastest to program without error, or the fastest to execute. The programmer may still program structured if he desires. He may even load a package to ensure this. And if the code definition is long then the statement for high level would apply also. The code should be well scrutinized as to reason.

Note that using SP and ?CSP is a simple and effective way to catch many of the errors made in either case. They can even be used outside of the definitions of : and ; or CODE and END-CODE, and never otherwise interfere. I am not saying there should be NO compiler security at all. If I had this viewpoint I would not have bothered to write the article. Even I feel much more comfortable with an application that I have programmed after it has been successfully loaded into a secure system. But I do object to having to program around the compiler security (which wastes time and introduces errors) when I wish to load a perfectly correct program which the security does not like. With an optional package, I can check my application as desired, but do not have to fight the compiler security to get the job done. Or, I can have the security package resident. I make the choice.

George W. Shaw
SHAW LABS LTD.
PO Box 303
San Lorenzo, CA 94580

FIG Convention Coming — Nov. 28

Here is a very short contribution, a compiler extension, which has been helpful to me. I want to share it with all FIGgers; perhaps it can become a fig-FORTH standard.

Often in creating a definition, we want to test or output an ASCII character, using words like EMIT or = or possibly even in a CASE statement. These are normally supplied as literal numbers in the current radix. These compile into the usual dictionary pair of LIT followed by the literal value. The difficulty is that we must either determine the ASCII code experimentally beforehand, or else reach for some reference list (usually in the wrong radix).

This compiler extension allows any editor-acceptable character to be displayed in its real form while compiling into a normal literal pair. While this may prove to be a minor help at edit-time, the resultant source code is much more readable at a later time, and is self-commenting, both highly desired features of any programming language.

The new word is ASCII, and it is followed by a literal character. The definition of ASCII is simple:

```
: ASCII BL WORD HERE 1+ C@
    [COMPILE] LITERAL; IMMEDIATE
```

It is made immediate so that it executes during compile-time. WORD takes the next input-stream text, delimited by a blank, and places it at HERE. Then the first character is placed on the stack for use by LITERAL, which has been forced to be compiled into our definition. What could be easier?

Formerly, we had to write 65 EMIT to output the letter "A" (assuming decimal radix). Now we can write ASCII A EMIT , clearly the better for everybody's understanding. The choice of the word ASCII is open to change, but the idea is a valuable addition to our efficient use of the language.

That's my contribution. I hope others can use it to improve their work. Thank you for providing a medium for ideas.

Raymond Weisling
Jln. Citropuran No. 23
Solo, Jawa Tengah
Indonesia

MULTIPLE WHILE SOLUTION

I have no way of knowing whether this solution to the multiple WHILE problem is generally known, though I am sure that many people must be using it. The note has been kept as short as possible, and could easily be expanded.

(: ENDWHILE 2 - ?COMP 2 ?PAIRS could be simplified to : ENDWHILE ?COMP 4 ?PAIRS probably--it weakens the ENDIF analogy a little.

Many of your readers may not be familiar with ENDWHILE as a means of achieving multiple WHILEs in a BEGIN loop. It is simple and convenient, but not elegant. ENDWHILE is used in the construction

```
BEGIN. . .(test) WHILE. . .(test) WHILE. . .
    ENDWHILE ENDWHILE AGAIN or

BEGIN. . .(test) WHILE. . .(test) ENDWHILE
    UNTIL
```

with one ENDWHILE for each WHILE in the loop.

The definition is

```
: ENDWHILE 2 - ?COMP 2 ?PAIRS HERE
    4 + OVER - SWAP ! ; IMMEDIATE.
```

It causes WHILE to compile a branch to the word following AGAIN or UNTIL, and is directly analogous to ENDIF (THEN). It can be easily understood by comparing the definitions of WHILE and IF, and ENDWHILE and THEN.

A similar ENDWHILE can be defined for use in the ASSEMBLER vocabulary.

The ENDWHILE construction is awkward (poor English) but simple, and is worth using until something better is decided on.

Julian Hayden
2001 Roosevelt Avenue
Vancouver, WA 98660

FORTH STANDARDS CORNER

Robert L. Smith

The word WORD has caused implementers of the 79-Standard a certain degree of difficulty. The definition of WORD as it appears in the FORTH-79 Standard is as follows:

WORD char --- addr 181

Receive characters from the input stream until the non-zero delimiting character is encountered or until the input stream is exhausted, ignoring leading delimiters. The characters are stored as a packed string with the character count in the first position. The actual delimiter encountered (char or null) is stored at the end of the text but not included in the count. If the input stream was exhausted as word is called, then a zero length will result. The address of the beginning of this packed string is left on the stack.

There are a number of problems with the definition as it stands. Later I will suggest a slightly modified definition which should clarify the apparent intent of the Standards Team, although some of the problems will remain for the present.

1. The phrase "non-zero delimiting character" presumably means that char must not be the null character. An error condition should be specified if char is found to be zero.

2. The character count is to be stored in the first character position of a packed string. That could mean that the character count could not exceed 127. Since a string holds a sequence of 8 bit bytes, the Clarification Committee of the Rochester Standards Conference felt that the term "character position" was a typographical error that should have been "byte position", thus allowing a string count up to 255 characters.

3. Since the source string could be as long as a block (1024 bytes), the character count could exceed 255. This case should be specified as an error condition. The action to be taken on an error condition depends on the implementation. A number of schemes have been proposed, but there are none that are completely satisfactory. Many people, including this author, feel that any count should be allowed.

4. The definition uses the phrase "actual delimiter encountered (char or null)". I do not believe that the Standards Team meant to required implementations to use a null as a universal delimiter, although many undoubtedly will. The sequence in which the above-mentioned phrase appears probably means that if the end of the input stream is encountered before the specified terminating character is seen, then a null should be appended at the end of the packed string instead of the specified terminating character.

5. Note that in addition to being a terminating delimiter, char also specifies initial characters to be skipped. That property makes WORD very difficult to use in conjunction with strings which may have a zero length. An example of a zero length string is the null comment ( ). If one attempts to use WORD in a straightforward manner to enclose the command terminated by the right parenthesis, he will find that it and all succeeding text will be skipped! Since under the Standard, the use of WORD is about the only way that one has access to the contents of the text input buffer, this limitation appears to this writer to be unreasonable.

I believe that the following definition of WORD meets the essential intent of the Standards Team, and clarifies the problems stated in (1-4). in order to not add to the confusion, I have put a new serial number on the definition.

WORD char --- addr 301

Receive characters from the input stream according to the delimiter char and place the characters in a string beginning at

addr+1. The character count is placed in the byte position at addr. An error condition results if char is an ASCII null or if the count exceeds 255. Initial occurrences of char in the input stream are ignored. If char appears in the input stream as a terminating character, it is appended to the string but not included in the count. If the input stream is exhausted before char is encountered as a terminating character, the terminating character null is appended instead of char. A zero length will result if the input stream is exhausted when WORD is called.

The problem of the character count limitation could be considered in the future. One simple approach would be to use a full word for the character count. Another would be to eliminate the character count and always append a null at the end. The user could then do his own scanning. The problem of null length strings could be "defined" away by making null length strings illegal. I think that that is a poor solution. The real problem is that WORD is poorly factored. As usual in FORTH, the less a word does, the more useful it becomes. The process of scanning for initial delimiters should be separated from the process of scanning for terminating delimiters.

NEW PRODUCT

Z8002 Software Development System under CP/M® or Cromemco CDOS. Includes cross compiler and a number of utilities. Available on 5¼" or 8" disks. Price: $4,000.00

Inner Access Corp.
517K Marine View
Belmont, CA 94002
(415) 591-8295

"Starting FORTH"
Available NOW!

$16.00 Paperback
$20.00 Hardcover

Mountain View Press
PO Box 4656
Mt. View, CA 94040
(415) 961-4103

THE FORTH-79 STANDARD--A TOOL BOX?

George W. Shaw, II

As a vendor of a version of the FORTH language, and a self-proclaimed priest of the FORTH religion (I carry a soap box in my back pocket and will mount upon it at a moments notice), I am very interested in the best standardization of the FORTH language possible. There are many items in the '79 standard which need work. Many cannot, and maybe should not, be changed this time around, but will have to wait for subsequent standardization efforts. To this end, I am conducting interviews to compile as complete a list of problem areas and solutions as possible. I would like to thank all of the people who have spent time giving me the input, comments and ideas which are the inspiration for this article.

Much discussion centers on the defining: "What is a '79 standard program?" Many of the questions are similar to "Can I xxx, and will it be standard?", or "My system has a zzz which does more than the standard says. Is it standard?". These are the wrong questions. Granted, many of these questions could be answered by more explanatory text within the standard. But, in general, the real question is "What does a standard mean?", or better "What is the FORTH-79 Standard?"

The '79 standard very clearly defines itself. But, unfortunately, it seems that many people skip reading the first page of the standard and branch right into the glossary. If one is to read the first page, one notices a section of great importance:

1. PURPOSE

   The purpose of this FORTH standard is to allow transportability of standard FORTH programs in source form among standard FORTH systems. A standard program shall execute equivalently on all standard FORTH systems.

This section very clearly states the standards purpose is "... to allow transportability of standard FORTH programs in source form..." Further, that the program "...shall execute equivalently..." The section previous to the above clarifies the extent:

0. FORWARD

The computer language FORTH was created by Mr. Charles Moore, as an extensible, multi-level environment containing the elements of an operating system, a machine monitor, and facilities for program development and testing.

States clearly of FORTH as "...containing elements of ..." the various environments. Thus, the standardized FORTH should be a language which contains only the elements necessary to allow the transportation and equivalent execution of programs between FORTH systems. This is even further limited by the definition of transportability.

transportability

This term indicates that equivalent execution results when a program is executed on other than the system on which it was created. See 'equivalent execution'.

Which implies that a '79 standard system (in this case, a system which contains only the standard words) does not necessarily allow program creation (development). This is not to say that one could not define within the standard the additional tools necessary to develop programs. Only that the set of standard words may not be sufficient for development. (The additional words necessary for development is definitely an area to be looked at for the next standard.)

Considering the above definitions, I propose this answer to the title question of this article: The FORTH-79 Standard is to be a basic tool box upon which other devices can be built. From the definitions within the standard one should be able to build almost any other needed tool or application. We do not know yet if this is the case. It is extremely unlikely that the initial effort would have encompassed all design possibilities. The '79 standard is a first effort--a place to start from; a base from which we may begin to determine the minimum additions necessary to allow all tools or applications to be built transportably.

Yet, even with this understanding, it may be felt that the standard is incomplete. In a few cases this may definitely be true. A good example of this is in the text dealing with the vocabulary mechanism. The standard seems extremely limiting and impossible to deal with. But, the solution is simple. Do as you have always done in FORTH. If a structure is inadequate for an application, define a structure which is adequate. The standard itself, by content, forces development in those areas which have not yet been fully developed. It forces new ideas, better solutions, and, hopefully, a better standard next time around by its own proper usage.

As for the two most asked questions mentioned earlier, read the standard carefully. Does it specifically or implicitly prohibit xxxing? If not, try to transport it to other systems. If you are still unsure, send the question to FIG, we'll work out a clarification and recommend it to the standards team. What if your system does more than zzz says? Can it be made to do only what zzz says by possibly not exercising options? If so, it is probably standard. Still not clear? Send in the questions. We need them to make a better FORTH-79 Standard document.

There are areas of the tool box which may be cluttered by parameter testing or unnecessary words. Some areas may require better factoring. Much work has yet to do done. These areas need to be exposed. Write FIG about them. All input is greatly appreciated. I have found that each person sees different valid problems. Many are seen by all, but most people usually see at least one that has not been seen before; an application or solution which had not been considered.

When considering the FORTH-79 Standard, treat it as a basic tool box. Additional tools are applications from the point of view of the standard. Extend it as necessary. Can you add what you need by defining it only in terms of standard words? If not, what is the minimum necessary to allow you to do that. More definitions or more explanations? Experience is all that will tell. Send in your results.

George W. Shaw, II
Shaw Laboratories, Ltd.
P. O. Box 303
San Lorenzo, CA 94580

Book Review:

title:	Starting FORTH
Author:	Leo Brodie
	FORTH, Inc.
Copyright:	1981
Publisher:	Prentice-Hall, Inc.
Price:	$19.95 (hardbound)
	$15.95 (softback)
Availability:	Mid September, 1981
Review by:	George W. Shaw II
	Shaw Laboratories, Ltd.

In most books the useful information begins in chapter one, or later. Starting FORTH is an exception. Useful information starts in the Forward section of the book.

The book is designed to be interactive. After only two pages of chapter one, you are typing at the terminal. It is seldom that a sentence will leave you thinking, "Now, what does that mean?" Analogies are used throughout. Any "buzz" words, or differences between systems, or phrases which might cause confusion are footnoted to explain in more detail. This presents extremely basic or nonessential ideas outside of the main text, allowing the book to be simple enough for the beginner, but not to become tiresome to the more knowledgeable. For example, in the sections of the book dealing with math, separate sections or footnotes are presented to explain what integers are or what an absolute value is (for beginners). Or to give additional information about a faster algorithm than was used in an example in the main text (for experts). Where appropriate, quizzes or exercises are interspersed within the chapters to help with understanding the material presented.

The book is written for the current "close-to-79-STANDARD" version of polyFORTH with notations or footnotes to indicate and explain the differences from the standard. Throughout the book, tables and lists are used to summarize and clarify the information presented. The occasional tables of new words (in glossary form) are of great help. They prevent having to dig through the text for the words to perform the practice problems. At the end of each chapter is a complete glossary of the new words. Also, at the end of each chapter are problems, with the answers in the appedicies. There are even a few surprise questions to lighten the air.

Moving from the general to the specific, the value of this work becomes even more apparent in the following chapter by chapter review.

The Introduction is not just one introduction, but two: one for beginners (to computers) and one for professionals. The beginnner's section explains conceptually what computers and computer languages are, using an analogy (as will often be found) to simplify. the professional's section answers the usual skeptical questions of "What is" and "Where is" Forth with an impressive list of facts about the language and applications in which it has been used.

Chapter one, "Fundamental Forth", presents the basic concepts of dictionary extensibility and problem definition immediately, so that after only two pages, you are typing at a terminal executing commands and defining words. The text steps the reader through the complete development of a program and then illustrates its execution with the previously mentioned cast. The operation of the stack is then illustrated, and the format of glossary entries explained.

Chapter two, "How to Get Results", presents the basic four arithmitic operations, calculator and definition style, with conversions between infix and postfix notations. Practice problems and stack pictures are provided to ensure comprehension. The next half of the chapter covers the basic single and double precision stack operations with excellent stack pictures and quizzes to help along the way.

Chapter three, the "Editor (and Staff)", again looks at the dictionary, but in terms of redefinition and FORGETing of words. Forth's use of the disk is also described, along with LISTing, LOADing, and the word "(" for comments.

Chapter four, "Decisions, Decisions, . . .", illustrates the IF ELSE THEN structure of Forth; the various conditional tests, their uses and alternatives; and flags and how to manipulate them.

Chapter five, "The Philosophy of Fixed Point", expands upon the basic four arithmetic operations with some of the composite (1+, 2+, etc.) and some miscellaneous operations. The operators for the return stack are introduced with examples of their use in ordering parameters for formula calculations. A discussion of benefits of floating or fixed point math is followed with instruction about scaling in fixed point to eliminate the need for floating point. Discussed also are the use of 32 bit intermediate operators and the use of rational approximations in fixed point.

Chapter six, "Throw It For a Loop", discusses the operation of the various types of loops in Forth. A new cast of characters illustrate the "how" of DO LOOPs, nesting loops, using IF ELSE THEN inside loops, etc. BEGIN UNTIL and BEGIN WHILE REPEAT are also introduced.

Chapter seven, "A Number of Kinds of Numbers", is divided into two sections: for beginners and for everyone. The beginners section gives an excellent tutorial introducing the novice to computer numbers. This section describes in detail both signed and unsigned single and double length numbers. Also covered are arithmetic shifts, bit-wise operations, number bases and ASCII character representation. The section for everyone explains Forth's handling of signed and unsigned single and double length numbers for input, formatted output and mathematical operations. The effect of BASE on I/O, some usage hints, and mixed operations are discussed.

Chapter eight, "Variables, Constants, and Arrays", discusses the uses and operation of these structures. Both single and double length structures are introduced. Example problems are used to show various designs for byte and single length arrays. Factoring definitions is also discussed.

Chapter nine, "Under the Hood", presents a very clear, detailed, explanation of the various types of execution and structures within a Forth system. Of the many things examined are: text interpretation, ticking ('), compiling, vectored execution, dictionary structure, colon definition execution, vocabularies, the Forth memory map and its pieces. Much of the detail applies to polyFORTH, but the theory is sufficiently general to apply to the operation and structure of most Forth systems.

Chapter ten, "I/O and You", discusses string and text manipulation as they relate to disk and terminal I/O. Block buffer and terminal buffer access is discussed with notes for multi-user systems. String operators and string to number conversion are also covered.

Chapter eleven, "Extending the Compiler: Defining Words and Compiling Words", weans the reader from the friendly cast of characters as it shows the code behind the faces. All of the aspects of Forth compiler are discussed including: time periods, the various compilers inside Forth, DOES> words and immediate words. D-charts are introduced.

Chapter twelve, : "Three Examples", presents three programming problems and their solutions as an example of good Forth style. Text manipulation is presented with a random paper generator; Data manipulation with a file system; and fixed point number manipulation with a math problem which would seem to need floating point.

Following chapter twelve are four appendices which contain the answers to the problems, the features of polyFORTH not discussed in the text, the differences from the '79 Standard and a summary index of the Forth words presented in Starting Forth.

On the whole, Starting Forth is very well organized and presented. On occasion a few topics seemed to appear out of nowhere, as the section on Factoring Definitions in the chapter about variables, constants, and arrays. But, these digressions only serve as short breaks from the subject at hand and do not detract from the organization of the material. The text is very complete and easily understood. I rate the book very highly for both the novice and intermediate Forth programmer.

THE FORTH ENGINE

David Winkel

What can computer architects do to make their lives interesting?

It has been clear for some time that building conventional Von Neumann computers is useful but dull. This in spite of large vendors' advertising literture which breathlessly announces new architectural advances for their latest machines. Meyers' book[1] has an entertaining discussion of the history of these "new" advances. For example, virtual storage goes back to the Atlas system (U. Manchester, 1959).

How can we improve performance? It appears that there are two practical ways:

a. Engineering - faster components, pipelines, caches, etc., applied to conventional architectures.
b. Architectural - building fundamentally different computers.

The engineering approach has been remarkably successful as shown by Seymour Cray's products. These machines do an excellent job with Fortran, but conceal gaps that programmers have adjusted to and, in fact, accept as theological necessities. For example, the array is a fundamental concept of Fortran, yet is only indirectly supported in hardware. Subscripts going out of range is a common run time error but the hardware happily goes on with the wrong data pointed to by a bad subscript.

The architectural approach would reverse the procedure. Build hardware to support a language. We can do this at several levels, the lowest being language-directed design where hardware features are added to support specific language features. An example would be Burroughs' concept of data descriptors to provide run time checking of subscript ranges. Another example would be a P-code machine. P-code is language-directed since it was proposed as an ideal machine for compiled Pascal. It would be less suited for FORTRAN for example. The general idea in language-directed design is to mirror important high-level language concepts in hardware. Semantic Gap is defined as the degree to which language features are not mirrored in hardware. Thus, the semantic gap for ALGOL running on a Burroughs B6500 would be small, for PL/1 running on CDC machines quite large.

If we reduce the semantic gap to zero, we have a direct execution machine where hardware mirrors all the constructs (both data and control) of the language. Good discussions and bibliographies are given in references 1 and 2.

Now we have the maximum in speed and the minimum of generality. The computer now runs only one language. What that language should be is a central question. The SYMBOL computer was an early, truly heroic, system built by Fairchild to directly execute the Symbol language.[3] This is a PL/1-like language with a great deal of power. System performance was spectacular and yet the entire exercise cannot be considered successful. A large part was due to language complexity which translated into hardware complexity. It was difficult to fix bugs and impossible to add features inadvertently left out.

What we need is a well-tested, simple language before we build a corresponding direct execution machine. FORTH is the obvious choice.

The goal of this research is to build the world's fastest FORTH engine. This is a no-compromise effort to force the hardware to mirror the language. We did not start by saying it must be built with bit slices, or PLA's, or ... In fact, an early paper design was done with bit slices and discarded because it was too slow.

The measure of speed is clock cycles per instruction. Clock rate, in turn, is a function of technology, not architecture. The machine currently runs at 333 ns but could be easily speeded up by using ECL or Schottky logic and faster memories.

The design cycle for a FORTH primitive proceeds as follows:

a. Pick a primitive such as DO or LOOP.

b. Postulate hardware data paths, stacks, registers, etc., that implement DO and mating primitives such as LOOP and +LOOP.

c. Make sure this hardware supports hidden logical concepts--in this case, I, J, K--and violates no other FORTH concepts.

d. Count clock cycles.

e. Repeat b-d until you can think of no more speedup possibilities.

f. Make an engineering choice for implementation. DO takes 2 clock cycles without overlap, 1 with. LOOP and +LOOP take 1 clock cycle. For the first machine, we use a 2-cycle DO and reserve the 1-cycle version for later enchancements. As a byproduct of this implementation we can support loop nesting to a depth of 1024.

This process is repeated for each FORTH primitive. Finally, this collection of individually optimized hardware must be forged into a coherent whole that makes engineering sense. The result[4] is not too surprising. There are data and return stacks plus separate stacks for loop control. Of course, the loop stacks are invisible to the programmer. An arithmetic unit operates from the data stack, etc. What is surprising is the mass of data paths required to support parallel operations such as 2SWAP in one clock cycle. The results are impressive. For the fig-FORTH primitives all but 4 can be executed in one or two clock cycles with the exception of multiply and divide which take 1 clock cycle per bit. The machine currently has 16k X 16 main memory with 1k X 16 stacks both extendable by 4 X. I/O is done with a slave 6809 with programmed access to the data stack and DMA access to main memory. Control is microprogrammed with a 2910 driving a 1k X 60 bit writeable control store. This follows Logic Engine [5] philosophy so the user has very pleasant access to the micromemory for tailoring high-speed special purpose instructions.

Results for randomly chosen instructions are given below. All comparisons are based on a 1 MHz 6809 running fig-FORTH. The FORTH engine runs at 3 MHz.

DUP	99 X faster	SWAP	132 X faster
@	101 X faster	U*	96 X faster
!	114 X faster	ROT	624 X faster
AND	126 X faster	DO...LOOP	110 X faster (null body)

As a rule of thumb the speedup is a factor of 100. Why the 6809 (or any other computer) is so slow is an interesting question and will be treated in a more formal paper.

We have received a number of inquiries about machine availability. Does anyone really need a machine this fast? It is obviously a large (200+ IC) machine in the minicomputer class and will cost more than a Z80. I would appreciate hearing from readers about this as well as memory and I/O requirements.

David Winkel
2625 Solar Drive #5
Salt Lake City, UT 84117

REFERENCES

[1] Advances in Computer Architecture, Glenford J. Meyers, Wiley, 1978.

[2] Southcon Conference, Atlanta, Georgia, 1981, Session 20/2, Phillip Crews.

[3] SYMBOL - A Major Departure from Classic Software Dominated Von Neumann Computing Systems. Proc. 1971 Spring Joint Computer Conf., AFIPS, 1971, pp. 575-587.

[4] Southcon Conference, Atlanta, Georgia, 1981, Session 20/4, David Winkel

[5] The Art of Digital Design, D. Winkel & F. Prosser, Prentice Hall, 1981.

THE FORTH, INC. LINE EDITOR

S. H. Daniel
System Development Corporation
500 Macara Avenue
Sunnyvale, CA 94086

The upcoming publication of Starting FORTH, which is destined to become the "bible" of FORTH neophytes everywhere, provides an opportunity to upgrade the existing fig-FORTH line editor at a very small cost in time and effort.

There are at least two good reasons why this upgrade should be done. The first is standardization. A user of any version of fig-FORTH will be able to step up to a polyFORTH system and use the line editor. Conversely, FORTH, Inc. customers who try fig-FORTH will not have to learn to use a different editor.

The second reason for adopting the polyFORTH editor is its increased flexibility and ease of use. The current fig line editor uses only the PAD for storage of user inputs for searches, deletions, and replacements. The polyFORTH editor employs both a FIND buffer and an INSERT buffer, in addition to the PAD. This allows both of the extra buffers to be loaded, and the contents reused several times, without extra typing by the user. This makes commands like D (Delete) and R (Replace) especially useful.

By taking a few hints from Starting FORTH, and combining them with the existing editor, I was able to write a line editor which is functionally identical to the polyFORTH editor, but which is in the public domain and can be used by anyone.

SYSTEM REQUIREMENTS

This editor should run on any fig-FORTH system, including FORTH-79 Standard systems (if the changes mentioned in the section FORTH-79 Standard are made). The compiled line editor requires approximately 2K bytes of memory, plus room in the system for the PAD and the FIND and INSERT buffers. It operates within the confines of the default data and return stacks.

A high level version of the word MATCH, used by the line editor for searches, is included for those who do not already have a version written in assembly language. If you intend to use this version of MATCH, screens 216 and 217 should be loaded prior to loading the rest of the line editor. Credit for this version of MATCH goes to Peter Midnight of Hayward.

THE EDITOR COMMANDS

The word 'text' following a command indicates that any text typed after the command will be copied to the text buffer used by that command. The buffer contents will then be used when the command executes. If no text is typed by the user, the contents of that buffer (left over from the previous command or commands) will be used without modification in the execution of the command.

X eXtract (--)

Copies the current line into the INSERT buffer, and removes it from the screen. All following lines are moved up, and line 15 is left blank.

T Type (n --)

Type line n from the current screen. Set the cursor to the start of the line.

L List (--)

Like the FORTH word LIST, except that the current screen number is obtained from the variable SCR, rather than being typed in by the user.

N Next (--)

Increments the current screen number by one. This command is used just before the L command, to allow the user to list the next sequential screen.

B Back (--)

Decrements the current screen number by one. This command is also used before the L command, to allow listing of the previous sequential screen.

P Put (--)
P text

Any following text will be copied into the INSERT buffer. The INSERT buffer will be copied into the current line, replacing its previous contents. If the text consists of one or more blanks, the current line will be erased.

WIPE Wipe (--)

Erases the current screen. Equivalent to the original CLEAR command, except that the user need not enter the screen number.

COPY Copy (from -2, to-1 --)

Copy one screen to another.

F Find (--)
F text

Any following text is copied to the INSERT buffer. The contents of the INSERT buffer are compared to the contents of the current line. If a match is found, the line is displayed with the cursor positioned immediately after the end of the string searched for. The F command, with no following text, is exactly the same as the previous editor command N. If no match is found, the requested string is echoed to the terminal and the error message "NONE" is output.

E Erase (--)

Erases backwards from the cursor, according to the number of characters in the FIND buffer. This command should only be used immediately after the F command.

D Delete (--)
D text

Any following text is copied into the FIND buffer. The D command is a combination of the F and E commands. The string in the FIND buffer is matched against the contents of the current line, and if a match is found, the found string is deleted from the line.

TILL Till (--)
TILL text

Any following text is copied into the FIND buffer. Starting from the current cursor position, TILL searches for a match with the contents of the FIND buffer. If a match is found, TILL deletes all the text on the line from the current cursor position up to any including the end of the matched text.

S Search (last screen#+1 --)
S text

Any following text is copied into the FIND buffer. Starting at the top of the current screen and continuing until the bottom of the screen immediately before the screen number on the top of the stack, S searches for a match to the contents of the FIND buffer. Whenever a match is found, the line containing the match will be typed out, along with the line number and screen number in which the match occurred. Because of the way FORTH handles loops, the number on the top of the stack must be one higher than the highest screen to be searched.

I Insert (--)
I text

Any following text will be copied into the INSERT buffer. The I command copies the contents of the INSERT buffer into the current line, starting at the current cursor position. Any text to the right of the cursor will be pushed to the right and will be pushed off the line and lost if the total length of the line exceeds 64 characters.

U Under (--)
U text

Any following text will be copied into the INSERT buffer. Spread the screen at the line immediately below the current line, leaving a blank line. All following lines are pushed down. Any text on line 15 will be lost. The contents of the INSERT buffer will be copied into the blank line, and that line will be made the current line.

R Replace (--)
R text

Any following text is copied into the INSERT buffer. The R command operates as a combination of the E (Erase) and I (Insert) commands. Starting at the current cursor position, and working backwards towards the start of the line, text corresponding to the

length of the contents of the FIND buffer is deleted, and the contents of the INSERT buffer are inserted into the line. Since the contents of the FIND buffer determine how much text will be erased, the R command should only be used immediately following the F (Find) command.

M Move (Block#, Line# --)

Copies the current line into the INSERT buffer, then copies the INSERT buffer to the block, specified by Block#, UNDER the line specified by LINE#. The original block number is restored, and the next line in the block becomes the current line. This allows sequential lines to be moved with a minimum of keystrokes. One unfortunate side-effect of this command is that to move something to line 0 of another screen, you must first move it UNDER line 0, using the command xxx 0 M, make screen xxx current, and then extract the old line 0, moving everything else up.

↑ (--)

Used as a terminator for all commands allowing text input, such as P, F, R, etc. Allows more than one command to be entered on a single line, e.g.,

3 T P This is line 3↑ L (cr)

Although useful, this feature does preclude the use of the " " as a character in any text to be put on a screen.

GLOSSARY

The following glossary addresses all the FORTH words in the line editor except the actual editing commands, which are discussed above.

TEXT (delimiter --)

Any following text in the input stream, up to but not including the delimiter, is moved to the PAD. The length of the input string is stored at PAD, with the actual string starting at PAD+1. In FORTH-79 Standard systems, if no text follows in the input stream, a length byte of 0 will be stored. In non-Standard systems, a length byte of 1 will be stored, but PAD+1 will contain a null to indicate the absence of text.

(LINE) (Line#, Screen# -- Buffer Address,64)

Using the line and screen numbers, computes the starting memory address of the line in the disk buffer. May not be necessary in FORTH-79 Standard systems, depending upon implementation. Should already be present in earlier implementations.

LINE (Line# -- Buffer Address)

Ensures that the line number is within the legal range of the current screen, then uses (LINE) to set the starting address of the line in the disk buffer.

WHERE (Offset, Block# --)

Used when a compile time error occurs during loading. Converts the block number to a screen number, makes that screen current, and prints the line in which the load error occurred. Underneath the line in error, the cursor is printed to show the approximate location of the error. Enables the EDITOR vocabulary as it exists. Strictly speaking, this is not part of the polyFORTH editor, but it is a highly useful tool.

#LOCATE(--Cursor offset, line#)

Uses the current cursor position to compute the line number which contains the cursor and the offset from the beginning of the line to the current cursor position.

#LEAD (--Line address, offset to cursor)

Computes the beginning address of the current line in the disk buffer, and the offset from the start of the line to the current cursor position.

#LAG (-- cursor address, count after cursor)

Computes the address of the cursor in the disk buffer and the count of characters remaining on the line after the cursor.

-MOVE (from address, to line# --)

Moves a line within the disk buffer to the line specified, completely replacing the previous contents of that line.

?MOVE (destination buffer address --)

If any text has been entered into the PAD by TEXT, moves that text to the specified buffer. Used to load the FIND and INSERT buffers for searches, etc. If no text was in the PAD, no action is taken.

>LINE# (-- current line number)

Uses the stored cursor location to compute the current line number.

FIND-BUF (--)

Establishes the FIND buffer a fixed distance above the current address of the PAD.

INSERT-BUF (--)

Establishes the INSERT buffer a fixed distance above the FIND buffer.

(HOLD) (Line# --)

Non-destructively copies the contents of the current line to the INSERT buffer.

(KILL) (Line# --)

Replaces the specified line with a blank line.

(SPREAD) (--)

Spreads the screen, starting at the current line, pushing all lines below the current line down, and leaving the current line blank. Any text on line 15 is pushed off the screen and is lost.

DISPLAY-CURSOR (--)

Displays the current line with the cursor in place.

(R) (--)

Replaces the current line with the contents of the INSERT buffer. Used as the primitive operation for the P command.

(TOP) (--)

Resets the stored cursor position to the top of the screen.

1LINE (-- Flag)

Scans the current line for an exact match with the contents of the FIND buffer. If a match is found, the stored cursor position is updated.

(SEEK) (--)

Starting at the current cursor position, searches the rest of the current screen for an exact match to the contents of the FIND buffer. If no match is found, the contents of the FIND buffer are typed and the error message "NONE" is output.

(DELETE) (Count --)

Starting at the current cursor position, text is deleted backwards (towards the beginning of the line), according to the count. The remaining text on the line is moved to the left and the end of the line is filled with blanks.

(F) (--)

Copies any following text to the FIND buffer and searches the

current screen for a match. Used as the primitive operation for the F and D commands.

(E) (--)

Uses the length of the contents of the FIND buffer to set the count for (DELETE). Used as the primitive for the E and R commands.

COUNTER A variable used by the S command to count the number of lines output to the screen and printer.

BUMP (--)

Increments the number of lines output and sends a page eject when 56 lines have been output. Used by the S command to handle pagination on the console and printer.

FORTH-79 STANDARD

The following changes should be made to the load screens shown in order to bring the line editor into conformance with the FORTH-79 Standard:

Screen	Line(s)	Change
209	9,10	The FORTH word R should be changed to R@.
214	5,6,7	The FORTH word R should be changed to R@.
212	3	The 0 preceding the word VARIABLE should be deleted, since variables are initialized to 0 automatically under the FORTH-79 Standard.
202	12	The word 1+ may be deleted, since the FORTH-79 Standard specifies that a length byte of 0 will be stored when WORD finds no text in the input stream.

ERROR MESSAGES

Only two error messages are output by the line editor:

NONE Indicates that no match was found on the current screen corresponding to the contents of the FIND buffer.

NOT ON CURRENT EDITING SCREEN

Indicates that the line number passed to the word LINE was outside the legal range of 0-15 decimal.

```
SCR # 200
  0 ( polyFORTH compatible line editor                    810715 SHD )
  1
  2 FORTH  DEFINITIONS  HEX
  3
  4
  5 : TEXT              ( accept following text to PAD )
  6     HERE  C/L  1+  BLANKS  WORD
  7     HERE  PAD  C/L  1+  CMOVE  ;
  8
  9 : LINE              ( relative to SCR, leave address of line )
 10     DUP  0FFF0  AND
 11     IF  ." NOT ON CURRENT EDITING SCREEN"  QUIT  THEN
 12     SCR  @  (LINE)  DROP  ;
 13
 14 -->
 15
```

```
SCR # 201
  0 ( WHERE, #LOCATE                                          810707 SHD )
  1
  2 VOCABULARY  EDITOR  IMMEDIATE  HEX
  3
  4 : WHERE             ( print screen # and image of error )
  5     DUP  B/SCR  /  DUP  SCR  !  ." SCR # "  DECIMAL  .
  6     SWAP  C/L  /MOD  C/L  *  ROT  BLOCK  +  CR  C/L  TYPE
  7     CR  HERE  C@  -  SPACES  5E  EMIT
  8     [COMPILE]  EDITOR  QUIT  ;
  9
 10 EDITOR  DEFINITIONS
 11
 12 : #LOCATE          (  --- cursor offset-2, line-1 )
 13     R#  @  C/L  /MOD  ;
 14
 15 -->

SCR # 202
  0 ( #LEAD, #LAG, -MOVE, BUF-MOVE                            810707 SHD )
  1
  2 : #LEAD             (  --- line address-2, offset to cursor-1)
  3     #LOCATE  LINE  SWAP  ;
  4
  5 : #LAG              (  --- cursor adr-2, count after cursor-1)
  6     #LEAD  DUP  >R  +  C/L R>  -  ;
  7
  8 : -MOVE             ( move from adr-2, to line-1 ---  )
  9     LINE  C/L  CMOVE  UPDATE  ;
 10
 11 : BUF-MOVE         ( move text to buffer-1, if any  --- )
 12     PAD  1+  C@
 13     IF  PAD  SWAP  C/L  1+  CMOVE
 14     ELSE  DROP
 15     THEN  ;            -->

SCR # 203
  0 ( >LINE#, FIND-BUF, INSERT-BUF                            810707 SHD )
  1
  2
  3 : >LINE#            ( convert current cursor position to line#)
  4     #LOCATE  SWAP  DROP  ;
  5
  6
  7 : FIND-BUF          ( buffer used for all searches )
  8     PAD  50  +  ;
  9
 10
 11 : INSERT-BUF        ( buffer used for all insertions )
 12     FIND-BUF  50  +  ;
 13
 14 -->
 15

SCR # 204
  0 ( (HOLD-, (KILL-, (SPREAD-, X                             810707 SHD )
  1
  2 : (HOLD)            ( move line-1 from block to insert buffer )
  3     LINE  INSERT-BUF  1+  C/L  DUP  INSERT-BUF  C!  CMOVE  ;
  4
  5 : (KILL)            ( erase line-1 with blanks )
  6     LINE  C/L  BLANKS  UPDATE  ;
  7
  8 : (SPREAD)          ( spread, making line# blank )
  9     >LINE#  DUP  1  -  0E
 10     DO  I  LINE  I  1+  -MOVE  -1  +LOOP  (KILL)  ;
 11
 12 : X             ( delete line# from block, put in insert buffer)
 13     >LINE#  DUP  (HOLD)  0F  DUP  ROT
 14     DO  I  1+  LINE  I  -MOVE  LOOP  (KILL)  ;
 15 -->

SCR # 205
  0 ( DISPLAY-CURSOR, T, L                                    810715 SHD )
  1
  2
  3 : DISPLAY-CURSOR    (  --  )
  4     CR  SPACE  #LEAD  TYPE  5E  EMIT
  5     #LAG  TYPE  #LOCATE  .  DROP  ;
  6
  7
  8 : T                 ( type line#-1 )
  9     C/L  *  R#  !  0  DISPLAY-CURSOR  ;
 10
 11
 12 : L                 ( list current screen )
 13     SCR  @  LIST  ;
 14
 15 -->
```

```
SCR # 206
  0 ( N, B, (TOP-, SEEK-ERROR                                  810707 SHD )
  1
  2 : N                   ( select next sequential screen )
  3     1  SCR  +!  ;
  4
  5 : B                   ( select previous sequential screen )
  6     -1  SCR  +!  ;
  7
  8 : (TOP)               ( reset cursor to top of block )
  9     0  R#  !  ;
 10
 11 : SEEK-ERROR          ( output error mssg if no match )
 12     (TOP)  FIND-BUF  HERE  C/L  1+  CMOVE
 13     HERE  COUNT  TYPE
 14     ."  NONE"  QUIT  ;
 15 -->


SCR # 207
  0 ( (R-, P                                                   810707 SHD )
  1
  2
  3 : (R)                 ( replace current line with insert buffer )
  4     >LINE#
  5     INSERT-BUF  1+  SWAP  -MOVE  ;
  6
  7
  8 : P                   ( following text in insert buffer and line)
  9     5E  TEXT
 10     INSERT-BUF  BUF-MOVE
 11     (R)  ;
 12
 13
 14 -->
 15


SCR # 208
  0 ( WIPE, COPY, 1LINE                                        810715 SHD )
  1
  2
  3 : WIPE                ( clear the current screen )
  4     10  0  DO  I  (KILL)  LOOP  ;
  5
  6 : COPY                ( copy screen-2 onto screen-1 )
  7     B/SCR  *  OFFSET  @  +  SWAP  B/SCR  *  B/SCR
  8     OVER  +  SWAP
  9     DO  DUP  FORTH  I  BLOCK  2  -  !  1+  UPDATE  LOOP
 10     DROP  FLUSH  ;
 11
 12 : 1LINE      ( scan current line for match with FIND buffer )
 13              ( update cursor, return boolean                 )
 14     #LAG  FIND-BUF  COUNT  MATCH  R#  +!  ;
 15 -->


SCR # 209
  0 ( (SEEK-, (DELETE-                                         810715 SHD )
  1
  2 : (SEEK)      ( FIND buffer match over full screen, else error)
  3     BEGIN  3FF  R#  @  <
  4       IF  SEEK-ERROR  THEN
  5       1LINE
  6     UNTIL  ;
  7
  8 : (DELETE)            ( backwards at cursor by count-1 )
  9     >R  #LAG  +  R  -           ( save blank fill location )
 10     #LAG  R  MINUS  R#  +!      ( back up cursor )
 11     #LEAD  +  SWAP  CMOVE
 12     R>  BLANKS  UPDATE  ;       ( fill from end of text )
 13
 14   -->
 15
```

```
SCR # 210
  0 ( (F-, F, (E-, E                                        810715 SHD )
  1
  2 : (F)                 ( find occurance of followins text )
  3     5E  TEXT
  4     FIND-BUF  BUF-MOVE
  5     (SEEK)  ;
  6
  7 : F                   ( find and displaw followins text )
  8     (F)  DISPLAY-CURSOR  ;
  9
 10 : (E)                 ( erase backwards from cursor )
 11     FIND-BUF  C@  (DELETE)  ;
 12
 13 : E                   ( erase and displaw line )
 14     (E)  DISPLAY-CURSOR  ;
 15 -->


SCR # 211
  0 ( D, TILL                                               810715 SHD )
  1
  2
  3 : D                   ( find, delete, and displaw followins text)
  4     (F)  E  ;
  5
  6
  7 : TILL                ( delete from cursor to text end )
  8     #LEAD  +  5E  TEXT
  9     FIND-BUF  BUF-MOVE
 10     1LINE  0=  IF  SEEK-ERROR  THEN
 11     #LEAD  +  SWAP  -  (DELETE)
 12     DISPLAY-CURSOR  ;
 13
 14 -->
 15


SCR # 212
  0 ( COUNTER, BUMP                                         810707 SHD )
  1
  2
  3 0  VARIABLE  COUNTER
  4
  5
  6 : BUMP                ( the line number and handle rasins )
  7     1  COUNTER  +!    COUNTER  @
  8     38 >  IF    0 COUNTER !
  9     CR  CR  0F  MESSAGE  0C  EMIT  THEN  ;
 10
 11 -->
 12
 13
 14
 15


SCR # 213
  0 ( S                                                     810715 SHD )
  1
  2 : S                   ( from current to screen-1 for strins )
  3     0C  EMIT  5E  TEXT   0  COUNTER !
  4     FIND-BUF  BUF-MOVE
  5     SCR  @  DUP  >R  DO  I  SCR  !
  6     (TOP)
  7     BEGIN
  8         1LINE  IF  DISPLAY-CURSOR  SCR  ?  BUMP  THEN
  9         3FF  R#  @  <
 10     UNTIL
 11     LOOP  R>  SCR  !  ;
 12
 13 -->
 14
 15
```

```
SCR # 214
  0 ( I, U                                                    810715 SHD )
  1 : I                   ( insert text within line )
  2     5E  TEXT                        ( load insert buffer with text)
  3     INSERT-BUF  BUF-MOVE            ( if any )
  4     INSERT-BUF  COUNT  #LAG  ROT  OVER  MIN  >R
  5     R  R#  +!                       ( bump cursor )
  6     R  -  >R                        ( characters to save )
  7     DUP  HERE  R  CMOVE             ( from old cursor to HERE )
  8     HERE  #LEAD  +  R>  CMOVE       ( HERE to cursor location )
  9     R>  CMOVE  UPDATE               ( PAD to old cursor )
 10     DISPLAY-CURSOR  ;               ( look at new line )
 11
 12 : U                   ( insert following text under current line)
 13     C/L  R#  +!  (SPREAD)  P  ;
 14
 15 -->


SCR # 215
  0 ( R, M                                                    810715 SHD )
  1
  2 : R                   ( replace found text with insert buffer )
  3     (E)  I  ;
  4
  5 : M                   ( move from current line on current screen )
  6     SCR  @  >R        ( to screen-2, UNDER line-1 )
  7     R#  @  >R         ( save original screen and cursor location )
  8     >LINE#  (HOLD)    ( move current line to insert buffer )
  9     SWAP  SCR  !      ( set new screen # )
 10     1+  C/L  *  R#    ! ( text is stored UNDER requested line )
 11     (SPREAD)  (R)     ( store insert buffer in new screen )
 12     R>  C/L  +  R#    ! ( set original cursor to next line )
 13     R>  SCR  !  ;     ( restore original screen )
 14
 15 FORTH  DEFINITIONS  DECIMAL


SCR # 216
  0 (                                                         810715 SHD )
  1 FORTH  DEFINITIONS  DECIMAL
  2 : 2DROP   DROP  DROP  ;       ( drop a double number )
  3
  4 : 2SWAP                       ( 2nd double number to TOS )
  5     ROT  >R  ROT  R>  ;
  6
  7 : 2DUP    OVER  OVER  ;       ( dup a double number )
  8
  9 : (MATCH)                     ( addr-3, addr-2, count-1 -- flag )
 10     -DUP  IF  OVER  +  SWAP
 11               DO
 12                   DUP  C@  I  C@  -
 13                   IF  0=  LEAVE  ELSE  1+  THEN
 14               LOOP
 15           ELSE  DROP  0=  THEN  ;                                -->


SCR # 217
  0 (                                                         810715 SHD )
  1
  2 : MATCH          ( cursor adr-4, bytes left-3, string adr-2 )
  3                  ( string count-1 -- flag-2, cursor offset-1 )
  4     >R  >R  2DUP  R>  R>  2SWAP  OVER  +  SWAP
  5     ( caddr-6, bleft-5, $addr-4, $len-3, caddr+bleft-2, caddr-1)
  6     DO
  7         2DUP  I  SWAP  (MATCH)
  8         IF
  9             >R  2DROP  R>  -  I  SWAP  -  0  SWAP  0  0  LEAVE
 10         ( caddr, bleft, $addr, $len    OR     0, offset, 0, 0 )
 11         THEN
 12     LOOP
 13     2DROP   ( caddr-2, bleft-1    OR    0-2, offset-1 )
 14     SWAP  0=  SWAP  ;
 15
```

RECURSION AND THE ACKERMANN FUNCTION

Joel V. Petersen

Recursion involves the calling of a program by itself. An example of where recursion might be used is in the parenthesis handler of an algebraic string parser. Every time the parser encounters a left parenthesis, it calls itself; every time the parser encounters a right parenthesis, it completes a call of itself. Recursion is somewhat difficult to explain and very difficult to use properly. However, the implementation of recursion in any language can be tested with a program called the Ackermann Function. This is a recursive function of two variables which is almost impossible to explain. The following is an implementation of the function in PASCAL.

```
VAR K,J: INTEGER; CALLCNT;INTEGER;

FUNCTION F(K,J: INTEGER): INTEGER;
BEGIN
   CALLCNT :=CALLCNT+1;
   IF K=0 THEN
        F :=J+1
   ELSE
        IF J=0 THEN
             F := F(K-1,1)
        ELSE
             F :=F(K-1,F(K,J-1));
END(*ACKERMANN FUNCTION*);
```

Recursive programming as illustrated in the PASCAL example is not possible in FORTH. A program can not invoke itself simply by using its own name while defining that word. However, recursion is not difficult at all to achieve:

```
(FIG-FORTH)
   : MYSELF LATEST PFA CFA , ;
   IMMEDIATE

(NIC-forth)
   : MYSELF LAST @ @ 2 + , ; IMMEDIATE
```

MYSELF simply places the address of the code field of the word being defined into its own definition. Thus, whenever the program needs to invoke itself, the word MYSELF should be used instead. The Ackermann Function now becomes:

```
(FIG-FORTH)
0 VARIABLE CALLCNT
: ACKERMANN ( I J - F )
   1 CALLCNT +!
   0= IF
        SWAP DROP 1+
   ELSE
        DUP
        0= IF
             DROP 1- 1 MYSELF
             ROT ROT DROP 1- SWAP MYSELF
        THEN
   THEN ;

(NIC-forth)

VARIABLE CALLCNT

: ACKERMANN ( I J - F )
   1 CALLCNT +! OVER
   THEN
        DUP
        THEN
             2DUP 1- MYSELF
             -ROT DROP 1- SWAP MYSELF
        ELSE
             DROP 1- 1 MYSELF
        ENDIF
   ELSE
        SWAPDROP 1+
   ENDIF ;
```

For comparison, the Ackermann Function was tested on the Nicolet 1280 20-bit processor in both (compiled) PASCAL and NIC-forth. The K=3, J=5 function took 8 seconds in (compiled) PASCAL and 12 seconds in NIC-forth. (As an aside, the addition of a simple hardware mod to the 1280 processor to speed up NEXT in NIC-forth reduced this to 9 seconds! Who says inline coding is so much faster than indirect threaded code!)

When attempting to try the Ackermann Function, one must allocate lots of room for both the parameter stack and the return stack. Every time the function is called, there must be two elements on the parameter stack, thus the parameter stack will fill up approximately twice as fast as the return stack. The K=3, J=6 function

requires over 1000 elements on the parameter stack and over 500 elements on the return stack at its deepest point. When the K=4, J=1 function was tried, the program finally crashed after five hours with the return stack containing over 5000 elements!!

The results of the simpler Ackermann Functions are given below. F is the value returned by the function. CALLCNT is the count of how many times the program called itself. MAXDEPTH is the maximum depth attained by the return stack.

K	J	F	CALLCNT	MAXDEPTH
0	0	1	1	
0	1	2	1	
1	0	2	2	
1	1	3	4	3
1	2	4	6	
2	0	3	5	
2	1	5	14	
2	2	7	27	8
2	3	9	44	10
2	128	259	33669	
3	0	5	15	
3	1	13	106	
3	2	29	541	
3	3	61	2432	63
3	4	125	10307	127
3	5	253	42438	255
3	6	509	172233	511
4	0	13	107	16
4	1	??	??	??

(NIC-forth is the implementation of FORTH on the NICOLET INSTRUMENT CORPORATION 1180/1280 series computers. This computer is a 20-bit minicomputer with a 19-bit address space.)

Joel V. Peterson
Nicolet Instrument Corp.
5225 Verona Road
Madison, Wisc. 53711
(608) 271-3333

(Ed. - A great article, but watch out. Most figFORTH implementations have insufficient stack space to execute this function. Programs should be reviewed for compatibility.)

REVIEW

A Brief Review of the Manuals
for the PET/CBM fullFORTH+ V1.3/4

by Jim Berkey

Complete system is available from IDPC Co., Box 11594, Philadelphia, PA 11916 for $65 (plus shipping?). Includes about 70 pages of documentation and a 5¼" diskette (not reviewed).

IDPC's fullFORTH+ is noted to have taken a person-year to be developed by an experienced programming staff. I give them a triple E for effort, but the product is, at best, rough.

fullFORTH+ is described as "a complete implementation of the FORTH language, as defined by the FORTH Interest Group." If this is true of the disk, then there are glaring technical errors in the glossary, whose definitions deviate substantially from the FIG manual. One example from +LOOP : "If the counter and limit values are equal, either before or after adding the increment, the DO loop is exited . . ." If you take this literally, the counter (read "index") is compared to the limit twice--once before and once after the increment--and exit can never occur on greater-than, as it does in the FIG model.

On the plus side, the package includes 6502 assembler, screen editor (not PET's), printer support, and floating point routines. These are nice to have, but from the samples of use shown, I suspect the presence of endless small inconveniences. To be fair, endless small inconveniences are a built-in feature of CBM disk systems which fullFORTH+ has not corrected.

I can't recommend fullFORTH+ for any but the desperate, because of two central problems: (1) the manual reveals a mangled view of the FIG model, and (2) fullFORTH+ was probably not implemented originally for the PET/CBM.

FORTH, INC. NEWS

BETTER SUPPORT PROMISED THROUGH FORTH, INC. AND TECHNOLOGY INDUSTRIES MERGER

FORTH, Inc. and Technology Industries, Inc. of Santa Clara, CA., have announced a merger. This means that FORTH, Inc. will become a wholly owned subsidiary of Technology, and the present shareholders of FORTH will become shareholders of Technology.

Technology Industries is a new company founded in February 1981 by John Peers. Peers is best known as founder and former chairman of Logical Machines Corp. of Sunnyvale, CA. This very successful company manufactures and sells business computers that feature a "programmerless" language called Adam, designed by Peers.

"The principle change that everyone will notice," said FORTH, Inc.'s president, Elizabeth Rather, "is that we'll be doing a lot more of what we do best--selling and supporting high quality professional FORTH systems and applications--and doing it even better. We're expanding our staff and investing heavily in equipment training."

FORTH, Inc. will operate with its individual identity, retaining the same name and operating structure. Technology Industries will be the "parent" of several other new companies as well. Each will specialize in hardware designed around and featuring FORTH. "Membership in this group will provide us with the opportunity to do some things I've wanted to do for years," said Chuck Moore. "I'm extremely excited about these plans."

EXPANSION CONTINUES

FORTH, Inc.'s growth in recent months has included two significant additions to management.

Joseph "Skip" Reymann, formerly with GOULD NAVCOM of El Monte, California, has joined FORTH, Inc. as vice president of operations. Reymann has extensive experience in both the technical and business aspects of program management. He has degrees in physics, finance, and corporate and contract law.

Robert E. Smith, Jr. is FORTH, Inc.'s new vice president of sales and marketing. Smith has over ten years of experience marketing application software for minicomputers. He has already tripled the size of the marketing department and plans to triple it again within eighteen months.

Other important additions to the staff include two people in the accounting department and three sales and marketing representatives. The products department has been reorganized with Leo Brodie, author of Starting FORTH, acting as manager. The publications department has grown by two, and three general support staff members have come on board.

RECENT APPLICATIONS

FORTH, Inc. recently signed a contract with International Business Services, Inc. in Washington, D.C., to supply hardware and software to the United States Forest Service.

FORTH, Inc. will provide the hardware and update and enhance the software for a high-resolution map analyzer system. The system will work with digitized data from existing contour maps in raster format.

The raster-scanned maps will be displayed on a high resolution (1024 x 1024) image system. A PDP-11/44 is then used to follow a given contour line and convert it to a string of vectors. Operator assistance is required in selecting a contour line, labeling, handling breaks in data, and making corrections from the original map. Operator input is via a track ball interface and alpha-numeric CRT.

Dick Liston of USFS has used FORTH for several years developing a prototype version of the system using miniFORTH on a PDP 11/05.

MARKETING COLUMN

Q. I've written several programs that all my friends think are excellent; what is the best way to market them?--M.L., New Mexico

A. There is no universally "best" way to market anything, and that includes computer programs. Generally speaking, however, planning is your best ally. Since you have already received some feedback (and I assume you are certain that it is valid and not just your friends being politely supportive), it makes sense that persons that closely match the profile of your friends in terms of need, occupation, income, etc. would be your best prospects. Simply put, marketing under these circumstances will consist of finding a way to communicate effectively and cost effectively with this target group.

Q. I've run a number of ads for software I have developed and while I have sold some, I just don't seem to make any real money for the time I am putting in--what am I doing wrong?--R.B., Sandusky, Ohio

A. Your problem points up many areas that do not occur to the amateur entrepreneur. In the interests of brevity, I will touch on a few of the more significant as being instructive to our readers.

* **PRODUCT**--in this area you may be promoting a product that serves no real need or is competing with an already established vendor.

* **PRICE**--your price may be too high, causing your potential customers to seek other sources or do without; or, more commonly, your price may be too low, causing you to perform excessive labor in selling and servicing your accounts for the amount you are charging.

* **MEDIA**--you may be advertising or selling to the wrong audience. If you have failed to research your market and are running ads based on who's cheapest as opposed to who's reading (prospect profile), you are unlikely to achieve any realistic sales. Remember your media should be purchased on the basis of cost per prospect, not cost per 1,000.

* **MESSAGE**--you may be saying the right thing to the right people, but in the wrong way. Part of your test marketing should be to give your advertising and sales copy to a rank amateur and see if what they think you are saying is the same thing you think you are saying.

The above list is by no means all-inclusive, but these are the areas you should start looking into first.

Q. Is there any way of selling my programs other than by buying ads, etc.?--B.C., Walnut Creek, CA

A. Yes. One of the most common ways is to have your software merchandised through any number of firms that specialize in this field. Basically the way they operate is to contract with your for ownership of your software and pay you a royalty on sales--much like an author receives from a book publisher. Naturally, the royalty is nowhere near the amount you would receive if you sold your software directly to the consumer yourself; but considering that you have no risk and your time is free to develop additional products which in turn can be sold, the reduced percentage is still often the best way to go. The point is that it isn't how large a percentage you receive that is important--but how much money you make.

Questions of general interest regarding the marketing of software will be answered in each edition in this column. Because of time limitations, it will not be possible to provide private answers either by phone or mail. In the interests of personal privacy, questioners will be identified by initials only. Questions should be addressed to:

MARKETING COLUMN
Editor, FORTH DIMENSIONS
PO Box 1105
San Carlos, CA 94070

HELP WANTED

FORTH PROGRAMMERS

Openings at All Levels
At FORTH, Inc.

Programmers experienced with mini/micro computers and peripherals to produce new polyFORTH systems and scientific/industrial applications. Degree in science or engineering and knowledge of FORTH essential.

PRODUCT SUPPORT PROGRAMMER

DUTIES: Responsible for maintaining existing list of software products, including polyFORTH Operating System and Programming Language, file management options, math options and utilities and their documentation; and providing technical support to customers.

Requirements include:

Good familiarity with FORTH--preferably through one complete target-compiled application. Assembler level familiarity preferred with the 8080, PDP/LSI-11, 8086, M6800, CDP1802, NOVA, IBM Series I, TI990. Communication skills are essential.

PROJECT MANAGER

Project manager to supervise applications and special systems programming projects: writing proposals, setting technical specifications, customer liaison, hands-on programming, and supervision of senior programmers.

SENIOR INSTRUCTOR

Experienced in course writing and development, technical education in computer software, hardware, and related subjects, including FORTH programming. Responsibilities include marketing seminar support and instructing in-house polyFORTH courses.

EDUCATIONAL STAFF ASSISTANT

Experienced in dealing with public, sales and marketing, and some programming. Duties will include assisting education department manager with overflow administrative tasks, active participation in FORTH, Inc. user group.

JR. INSTRUCTOR

Experienced in public speaking or educational instruction, programming on various processors --high-level languages and assembler. Microprocessor and FORTH programming background valuable.

CONTACT:

Pat Jones
FORTH, Inc.
2309 Pacific Coast Highway
Hermosa Beach, CA 90254
(213) 372-8493

CONSULTANT WANTED

We are designing a heat pump controller system, which is based on the National Semiconductor "COPS" Microcontroller. It is a 4 bit calculator chip, with 2K of ROM and 128 nibbles of RAM.

We need a consultant who can:

1. Advise whether or not Forth can be put on the COPS
2. Estimate the program size, once compiled
3. Write software which would allow me to write and debug code on a TRS-80, Model I, and then cross compile it to the COPS.

For information call:

THE COLEMAN COMPANY, INC.
Scott Farley
Design Project Manager
(316) 832-6545

NEW PRODUCTS

FORTH by Timin Engineering, Release 3

Release 3 of FORTH by Timin Engineering is a complete software development system. It is interactive (conversational) in nature. The FORTH system incorporates a command processor, compiler, editor and assembler, all memory resident. The principal benefits are a reduction in software development time and a reduction in memory size for large applications. The principal application area has been machine and process control. The language is suitable for all applications except scientific mathematics. This product is based on the well-known FIG FORTH but with numerous enhancements, including:

- visual (screen) editor
- array handling (implemented in machine code)
- very fast disk I/O
- configurable for different memory size
- creates turn-key applications
- CP/M system calls and file handling

Release 3 of Timin FORTH will run on Z-80/8080/8085 hardware systems with CP/M or CDOS. Minimum memory size is 28K. The price for Release 3 of Timin FORTH is $235 (if other than 8" standard disk, add $15). To order Release 3 of Timin FORTH, write Timin Engineering Company, 9575 Genesee Avenue, Suite E-2, San Diego, CA 92121, or call (714) 455-9008.

HDOS FORTH

- Vendor: Essex Computer Science

- Address: 1827 St. Anthony Ave., St. Paul, MN 55104

- Telephone: (612) 645-3345

- Contact: Rick Smith

- Product Name: Essex HDOS FIG-Forth

- Description:

  Essex HDOS FIG-Forth is an inexpensive version of FIG-Forth for Heath H89/Zenith Z89 users with the HDOS operating system. It is a version of 8080 FIG-Forth Version 1.1 customized for HDOS and the H/Z89. Disk I/O takes place via a standard HDOS disk file. In addition, the FIG-Forth source listings are provided and may be modified and re-assembled on a single-disk HDOS sytem.

- Extras: None.

- Target machines: Heath H89 and Zenith Z89. Heath H8 users may also use the system if they modify the console I/O routines.

- Memory requirements: 32K of RAM

- Number of documentation pages: 140

- Documentation description:

  Documentation consists of release notes, a copy of the FIG-Forth Installation Guide, and a copy of the official 8080 FIG-Forth version 1.1 source listing. The manuals provide the information necessary to install and modify the Forth system.

- Essex does not offer the manuals separately. They may be purchased separately through the Forth Interest Group.

- We will reduce the price to $25.00 for persons already owning copies of both FIG documents.

- Form of Product: 5" HDOS diskette, including source, object, and release note files.

- Shipments to date: about 4

- Price: $45.00, or $25.00 for those who already own the FIG documentation.

- Includes: U. S. postage, local tax.

- Warranties and support: 30 day free replacement of defective media. We are interested in fixing bugs that crop up but do not guarantee that bugs will get fixed.

- Order turnaround time: 3-4 weeks.

Order from:

Essex Computer Science
Richard E. Smith
1827 St. Anthony Avenue
St. Paul, MN 55104
(612) 645-3345.

AN 1802 FIG FORTH

Version 1--RCA CDOS
Load under RCA CDOS
Disc with source and object files for
RCA CDP18S008
CDP18S007
CDP18S005 with CDOS upgrade
A minimum of 8K from address 0 is required

Version 2--RCA unit-track
Load under RCA unit-track
Disc with source and object files for
RCA CDP18S008
CDP18S007
CDP18S005 with UART card
A minimum of 8K from address 0 is required

Version 3--object and FORTH screens
Load under RCA unit-track
then LOAD FORTH screens

Version 3 is suggested unless the user wants to manipulate the 1802 source code. This version will be continually updated with program material.

The discs are $50 each (Calif. res add 6 percent sales tax)
Order from: CMOSOFT, P. O. Box 44037, Sylmar, CA 91342

AIM-FORTH 'HACKER'S SYSTEM"

I finally got my fig-FORTH 65 running on my AIM-65 at work and I would like to offer it to other hackers like myself. This FORTH system runs on AIM -65 with the DAIN DISK SYSTEM and uses an external terminal.

The software is on 2 disks. One contains the complete source and object. The other contains Editor, Screens, Error Messages and other bits of FORTH code of my creation done while I started using FORTH.

I will supply my AIM-FORTH "Hacker's System" to anyone for $25.00. THIS IS NOT FOR BEGINNERS! THIS IS NOT A COMMERCIAL PRODUCT! I am interested in contacting other FORTH hackers in my area and would like to possibly make some noise with them or start a phone line software interchange of techniques using MODEMS. I welcome any letters or input on this idea.

Eric Johansson
55 A Richardson St.
Billerica, MA 01821
(617) 667-0137 (home)
(617) 899-2719 x 224 (work)

FORTH MAILING LIST FOR APPLE

Allows users to maintain 1,000 entries per floppy. Functions include adding, deleting, and modifying entries. The mount option allows mounting any number of mailing list floppys. Labels can be generated in 1,2,3, or 4 across formats with user optional selection criteria.

This application package includes: 16 sector boot disk for the Apple; Source code for system and a bonus of one mailing list floppy with name addresses and phone numbers of over 100 FORTH users.

Price is $45.00 from:

Elmer W. Fittery
INTERNATIONAL COMPUTERS
110 McGregor Avenue
Mt. Arlington, NJ 07856
(201) 663-1580 (call after 6:00 pm)

FIG CONVENTION COMING -- NOV. 28

FORTH CLASSES

NEW CLASS
BY KIM HARRIS & HENRY LAXEN

FORTH, PRINCIPLES AND PRACTICES

This class is intended to teach the student how to write programs in FORTH. It is a "how to" class and not a "why" workshop. The class will meet on each Monday in October from 6:30 to 9:30 at Berkeley Computer, 1569 Solano Avenue, Berkeley. The phone number there is 526-5600. The topics to be covered are:

The Language
Input Output Structure
String Handling
Vocabularies
Defining Words

This is an ambitious schedule, and depending on the level of the students, more or less will be covered. Experience with other computer languages would be helpful, though it is not required. There will be homework exercises, and machines will be available for students' use. For more information, contact Henry Laxen at (415) 525-8582.

SEMINARS, WORKSHOPS, CLASSES
FROM FORTH, INC.

Location	Seminar	Workshop
Los Angeles	October 15	October 16
San Diego	October 22	October 23

Introductory classes in polyFORTH programming will be offered September 14-18 and October 5-9 at FORTH, Inc. An advanced course will run October 12-16. Contact Kris Cramer for details. FORTH, Inc., 2309 Pacific Coast Highway, Hermosa Beach, CA 90254, (213) 372-8493.

MORE FORTH CLASSES

Intensive 5-day FORTH workshops are being offered at INNER ACCESS CORPORATION. These workshops provide an introduction to the FORTH programming language sufficient to design and debug programs to solve real problems. These workshops also serve to enhance one's understanding of the FORTH tools necessary for complex applications.

Workshop Dates	Time	Cost
Sept. 21-25	9-4:30	$295
Oct. 19-23		
Nov. 16-20		

To obtain more information on these workshops, call Inner Access (415) 591-8295 in Belmont (home of Marine World) in the San Francisco Bay Area.

AND MORE CLASSES

Free Beginner's Class for Apple users. In San Diego, two-session course on 9/26/81 and 10/30/81 at 1 p.m. at Computer Merchant, 5107 El Cajon Blvd. K. V. Amatneek, Instructor.

How to form a FIG Chapter:

1. You decide on a time and place for the first meeting in your area. (Allow at least 8 weeks for steps 2 and 3.)

2. Send FIG a meeting announcement on one side of 8-1/2 x 11 paper (one copy is enough). Also send list of ZIP numbers that you want mailed to (use first three digits if it works for you).

3. FIG will print, address and mail to members with the ZIP's you want from San Carlos, CA.

4. When you've had your first meeting with 5 or more attendees then FIG will provide you with names in your area. You have to tell us when you have 5 or more.

Northern California

4th Sat — FIG Monthly Meeting, 1:00 p.m., at Southland Shopping Ctr., Hayward, CA. FORML Workshop at 10:00 am.

Southern California

Los Angeles

4th Sat — FIG Meeting, 11:00 a.m., Allstate Savings, 8800 So. Sepulveda, L.A. Philip Wasson, (213) 649-1428.

Orange County

3rd Sat — FIG Meeting, 12:00 noon, Fullerton Savings, 18020 Brockhorst, Fountain Valley, CA. (714) 896-2016.

San Diego

Thur — FIG Meeting, 12:00 noon. Guy Kelly, (714) 268-3100, x 4784 for site.

Northwest

Seattle — Chuck Pliske or Dwight Vandenburg, (206) 542-8370.

New England

Boston

1st Wed — FIG Meeting, 7:00 p.m., Mitre Corp., Cafeteria, Bedford, MA. Bob Demrow, (617) 389-6400, x198.

Boston

3rd Wed — MMSFORTH Users Group, 7:00 p.m., Cochituate, MA. Dick Miller, (617) 653-6136 for site.

Southwest

Phoenix — Peter Bates at (602) 996-8398.

Tulsa

3rd Tues — FIG Meeting, 7:30 p.m., The Computer Store, 4343 So. Peoria, Tulsa, OK. Bob Giles, (918) 599-9304 or Art Gorski, (918) 743-0113.

Texas — Jeff Lewis, (713) 719-3320 or John Earls, (214) 661-2928 or Dwayne Gustaus, (817) 387-6976. John Hastings (512) 835-1918.

Mid Atlantic

Potomac — Joel Shprentz, (703) 437-9218.

New Jersey — George Lyons (201) 451-2905.

New York — Tom Jung, (212) 746-4062.

Midwest

Detroit — Dean Vieau, (313) 493-5105.

Foreign

Australia — Lance Collins (03) 292600.

England — FORTH Interest Group, c/o 38, Worsley Road, Frimley, Camberley, Surrey, GU16 5AU, England

Japan — FORTH Interest Group, Baba-bldg. 8F, 3-23-8, Nishi-Shimbashi, Minato-ku, Toyko, 105 Japan.

Canada

Quebec — Gilles Paillard, (418) 871-1960 or 643-2561.

West Germany

Wolf Gervert, Roter Hahn 29, D-2 Hamburg 72, West Germany,(040) 644-3985.

FORTH VENDORS

The following vendors have versions of FORTH available or are consultants. (FIG makes no judgment on any products.)

ALPHA MICRO

Professional Management Services
724 Arastradero Rd. #109
Palo Alto, CA 94306
(415) 858-2218

Sierra Computer Co.
617 Mark NE
Albuquerque, NM 87123

APPLE

IUS (Cap'n Software)
281 Arlington Avenue
Berkeley, CA 94704
(415) 525-9452

George Lyons
280 Henderson St.
Jersey City, NJ 07302
(201) 451-2905

MicroMotion
12077 Wilshire Blvd. #506
Los Angeles, CA 90025
(213) 821-4340

CROSS COMPILERS

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

polyFORTH

FORTH, Inc.
2309 Pacific Coast Hwy.
Hermosa Beach, CA 90254
(213) 372-8493

LYNX
3301 Ocean Park #301
Santa Monica, CA 90405
(213) 450-2466

M & B Design
820 Sweetbay Drive
Sunnyvale, CA 94086

Micropolis

Shaw Labs, Ltd.
P. O. Box 3471
Hayward, CA 94540
(415) 276-6050

North Star

The Software Works, Inc.
P. O. Box 4386
Mountain View, CA 94040
(408) 736-4938

PDP-11

Laboratory Software Systems, Inc.
3634 Mandeville Canyon Rd.
Los Angeles, CA 90049
(213) 472-6995

John S. James
P. O. Box 348
Berkeley, CA 94701

OSI

Consumer Computers
8907 LaMesa Blvd.
LaMesa, CA 92041
(714) 698-8088

Software Federation
44 University Dr.
Arlington Heights, IL 60004
(312) 259-1355

Technical Products Co.
P. O. Box 12983
Gainsville, FL 32604
(904) 372-8439

Tom Zimmer
292 Falcato Dr.
Milpitas, CA 95035

6800 & 6809

Talbot Microsystems
5030 Kensington Way
Riverside, CA 92507
(714) 781-0464

TRS-80

Miller Microcomputer Services
61 Lake Shore Rd.
Natick, MA 01760
(617) 653-6136

The Software Farm
P. O. Box 2304
Reston, VA 22090

Sirius Systems
7528 Oak Ridge Hwy.
Knoxville, TN 37921
(615) 693-6583

6502

Eric C. Rehnke
540 S. Ranch View Circle #61
Anaheim Hills, CA 92087

8080/Z80/CP/M

Laboratory Microsystems
4147 Beethoven St.
Los Angeles, CA 90066
(213) 390-9292

Timin Engineering Co.
9575 Genesse Ave. #E-2
San Diego, CA 92121
(714) 455-9008

Application Packages

InnoSys
2150 Shattuck Avenue
Berkeley, CA 94704
(415) 843-8114

Decision Resources Corp.
28203 Ridgefern Ct.
Rancho Palo Verde, CA 90274
(213) 377-3533

KV33 Corp.
PO Box 27246
Tucson, AZ 85726

68000

Emperical Res. Grp.
PO Box 1176
Milton, WA 98354
(206) 631-4855

Firmware, Boards and Machines

Datricon
7911 NE 33rd Dr.
Portland, OR 97211
(503) 284-8277

Forward Technology
2595 Martin Avenue
Santa Clara, CA 95050
(408) 293-8993

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

Zendex Corp.
6398 Dougherty Rd.
Dublin, CA 94566

Variety of FORTH Products

Interactive Computer Systems, Inc.
6403 Di Marco Rd.
Tampa, FL 33614

Mountain View Press
P. O. Box 4656
Mountain View, CA 94040
(415) 961-4103

Supersoft Associates
P.O. Box 1628
Champaign, IL 61820
(217) 359-2112

Consultants

Creative Solutions, Inc.
4801 Randolph Rd.
Rockville, MD 20852

Dave Boulton
581 Oakridge Dr.
Redwood City, CA 94062
(415) 368-3257

Elmer W. Fittery
110 Mc Gregor Avenue
Mt. Arlington, NJ 07856
(213) 663-1580

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

Inner Access
517K Marine View
Belmont, CA 94002
(415) 591-8295

Henry Laxen
1259 Cornell
Berkeley, CA 94706
(415) 525-8582

VENDORS SEE PUBLISHER'S COLUMN

FORTH INTEREST GROUP

NATIONAL CONVENTION

NOVEMBER 28, 1981

Marriott Hotel
Santa Clara, CA

9:00 am - 6:30 pm	Papers and Exhibits
7:30 pm	Dinner and Speakers

LEARN FORTH

How to learn and/or teach FORTH
and FORTH applications

Preregistration Form

Name(s) _____

Company _____

Address _____

City _____ State _____ Zip _____

Phone (____) _____ Ext _____

____ I am interested in presenting a paper on:

_____

____ Enclosed is a check for: #____ @ $3.00 admission(s) $_____

(100 Limit) #____ @ $20.00 dinner(s) $_____

Total $_____

____ FORTH Vendor: 8' Table @ $50.00 $_____

Return To: FORTH Interest Group
PO Box 1105
San Carlos, CA 94070