FORTH DIMENSIONS

DECEMBER 1978/JANUARY 1979 VOLUME 1 No. 4
(Published July 1979)

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

PUBLIC MEETINGS OF F.I.G

The Forth Interest Group is pleased to announce a public meeting series. We will meet on the fourth (!) Saturday of the month in Hayward, Ca., in the Special Events Room of the Liberty House Department Store, in the Southland Shopping Center (800 Southland Mall)

This room is on the third floor rear. The formal meeting begins at 1:00 PM, but we gather for lunch about 12 Noon.

The specific dates are July 28, Aug 25, Sept 22, Oct 27, and Nov 24. A single technical topic will be presented in detail, with workshop sessions on the fig-FORTH model, and any topics of immediate interest. In July, Dave Lyons will present the ascii memory dump of Forth which is part of his 6800 version. This Forth program shows the entire language map on one sheet of paper!

STAFF

The volunteer staffing of Forth Dimensions is a bit fluid. For this issue, our staff consisted of:

EDITOR	Bill Ragsdale
REVIEW	Dave Boulton
CONTRIBUTORS	Paul Bartholdi D. W. Borden
TYPESETTING	GLG Secretarial and Vydec
ARTWORK	Anne Ragsdale
CIRCULATION	6502 TIM and Persci

PUBLISHERS COMMENTS

The issues of Forth Dimensions have been scheduled for two month intervals, but have been at intervals determined by the overall activities of the steering committee. In the past months we have participated in the International Standards Team, given conference papers the Fourth West Coast Computer Faire, attended the Forth Users Meeting in Utrecht, Holland.

These events have been outwardly reflected in the large gap since Issue 3. For the near term, hope is in sight! We have Issue 5 ready for publication in three weeks, and Issue 6 should occur within a month. This will wrap up Volume 1. At this time we will evaluate our efforts, and plan for future activities. Member comment will be quite apropos during this period and will help shape future application of effort.

Thanks are due the membership for their patience during this period.

HISTORICAL PERSPECIVE

FORTH was created by Mr. Charles H. Moore in about 1969 at the National Radio Astronomy Observatory, Charlottesville, VA. It was created out of this dissatisfaction with available programming tools, especially for automation. Distribution of his work to other observatories has made FORTH the de-facto standard language 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.

FORTH enjoys a synergism of its features. It has none of the elephantine characteristics of PL/I or FORTRAN. It has a density and speed far surpassing BASIC, but retains an interactive nature during program development. Since it is extensible, special words are easily defined to give it the terseness of APL. Its clarity and consistency result from being the product of a single mind (as were APL and PASCAL).

Although the language specification and many implementations are in the public domain, many other implementations and application packages are available as program products of commercial vendors.

The Forth Interest Group is centered in Northern California, although our membership of 450 is world-wide. It was formed in 1978 by local Forth programmers to encourage use of the language by the interchange of ideas through seminars and publications. All effort is on a volunteer basis and the group is affiliated with no vendors.

;S FIG

THE "TO" SOLUTION

Paul Bartholdi
Observatoire De Geneve
CH-1290-Sauverny
Switzerland

At the Catalina standardization meeting, Chuck Moore suggested rapidly the "TO" construct to aleviate some, if not all, of the difficulties associated with address manipulation. This new construction seems to me extremely powerful. It should considerably decrease the number of errors and improve the readability of programs. It is very easy to understand and to use (much easier in fact than the constructs it is replacing!). Its implementation is also trivial but the lack of experience in using it may hide some difficulties. These notes try to sketch its potentials.

1. The " TO " concept

The basic concept is the following: The code associated with VARIABLEs is divided into two exclusive parts.

- The first one (or " fetch code ") is identical with the one associated normally with CONSTANT . It pushes on the stack the value (byte, word or words) in the parameter field.
- The second one (or " store code ") is new. It transfers the value (byte, word or words) from the stack into the parameter field, it is equivalent to <variable-name> ! (or C! or D! or F!).

The choice between the two codes depends on a state variable (which will be called %VAR hereafter) that has two possible states: Ø (or fetch state) and 1 (or store state).

If %VAR = Ø the first code is executed and %VAR is unchanged.

If %VAR = 1 the second code is executed and % VAR is immediately returned to Ø.

The operator " TO " sets %VAR to 1, forcing the next-called variable to store the content on top of the stack in its parameter field instead of fetching it.

2. Examples using " TO "

We suppose three single variables called A , B and C three double variables F , G and H , and three floating variables X , Y and Z . Then the following half-lines are equivalent. The first column assumes the old definitions of variables, the second one uses the new concept.

```
               OLD :                     NEW :

  i)   A  @  B  @  +  C  !         A  B  +  TO  C
 ii)   F  D@ G  D@ D- H  D!        F  G  D- TO  H
iii)   X  F@ Y  F@ F* Z  F!        X  Y  F* TO  Z
```

3. VARIABLEs or CONSTANTs ?

Some advantages of the new concept are quite evident from the previous examples. Just before the Catalina meeting, I came to the conclusion that variables should be dropped altogether. If A , B , ... Z had been constants in the previous examples then, using P and ! , the three lines would have been coded

```
  i)  A  B  +  '  C  !
 ii)  F  G  D- '  H  D!
iii)  X  Y  F* '  Z  F!
```

which is already better than the "old" above.

The difficulty starts with ARRAYs. If they push values onto the stack instead of addresses, then it becomes much more difficult to store values at the right places.

Note that ' C (or H or Z) takes two words but is really, at execution time, a single operator (LIT <address-of-C>). " TO " is then the shortest solution in terms of memory space, and more or less equivalent to the CONSTANT solution in terms of the time used.

4. The " address " problem

But the main advantage of " TO " in this context are the following:

- In the general sense a " CONSTANT " should be used as such, and never (or hardly ever) be changed. In particular it may reside in PROM . I suggest then to keep the constants as such, its associated code ignoring the value of %VAR . We then should redefine the variables to check %VAR and behave accordingly.

- One of the main unresolved points at Catalina was the definition and use of addresses for variables in the general sense. One consensus was obtained in September at the Geneva meeting, that is, as far as possible, addresses should be omitted. The " TO " concept solves this requirement admirably. No addresses are ever put on the stack, or manipulated explicitly. Then byte or word addressing is irrelevant. It is taken care of at the system level only, in the code of the second part (the "store code").

5. Portability

Because of this, FORTH programs become more portable. " TO " replaces all fetch and store operators which would or would not be distinct, which would work on byte or "position" or word addresses. Transporting a program from a micro to a large CDC implies now much less adaptations.

6. Clarity - security

Using less operators, in fact the strict minimum, is probably one of the best ways of improving clarity. Note also that " TO " appears as an infix operator to the programmer (and reader!). In terms of security, " TO " implies that only the parameter field of variables can be changed. Other addresses are not (at least directly) accessible.

This is certainly a tremendous gain for some environments.

7. The ARRAY problem

The use of " TO " with ARRAYs is not as simple as with variables, but still quite practicable. Note first that anything but a variable can stay between " TO " and the variable's name. If C is defined as an ARRAY (with the double associated code) then

```
A TO 4 C (instead of A @ 4 C ! )
DO I TO I C LOOP
(instead of DO I I C ! LOOP )
```

are quite alright and extend all the previously noted advantages of " TO ". But the programmer must take care that the index must not contain a variable. For example:

```
4 TO B ... A TO B C
```

will put the value of A into B instead of into C The necessary form should then be

```
A B TO C
```

which is surely less pleasant because of its asymmetry.

8. Fetching and storing inside a disk block

The problem extends of course to "virtual arrays" like the disk blocks. In this context, direct access should be considered separately from Data Management. The double code concept associated with " TO " should be extended to the Data Management operators. For the direct manipulation of data inside a disk block, I suggest the creation of a new operator, with the double code, associated with the form of

```
<relative-word-address>
<bloc-number> BLOC
```

(BLOC is of course just a provisional name!)

Example:

```
4 , 12 BLOC 5 + TO 5 12 BLOC
```

instead of

```
12 BLOCK DUP 4 + @ 5 +
SWAP 5 + !
```

or

```
12 BLOCK 4 + @ 5 + 12
BLOCK 5 + !
```

9. General access to the (whole) memory

No good FORTH programmer would ever accept to be strongly restricted in his access to the memory. But if the " TO " concept is really accepted, then all the @ and ! operators should disappear. I suggest, instead, to add (its usage could be restricted) a generalized array called MEMORY (or any equivalent) with once more the double code associated with it. Then <address> MEMORY would either fetch from or store into the real address.

As an example, we would have

```
Ø TO 15317 MEMORY instead of
Ø 15317 ! or

DO I MEMORY S. LOOP instead of
DO I @ S. LOOP etc.
```

MEMORY is clearly equivalent to @ and ! depending on $VAR.

10. Generalization of address matching: virtual arrays

The previous points 8 and 9 suggest a further generalization that may solve another problem we have had since the beginning of FORTH: the use of arrays as parameters for a procedure. It was generally solved by putting the address of the first element on the stack, and doing explicit address arithmetic in the procedure (see the FFT of Jim Brault for example). This is certainly neither clean nor fast.

What I propose is the following: A virtual array (VARRAY , DVARRAY , FVARRAY , CVARRAY etc.) behaves like an array, but does not reserve space, except for a pointer to the real array. The link between the virtual and any portion of the memory is established by the word MATCH .

For example:

```
1ØØ ARRAY CUSTOMER
1ØØ ARRAY STAR
    VARRAY NUMBER
  , CUSTOMER MATCH NUMBER
```

associates the real array CUSTOMER with the virtual array NUMBER .

Then <i> NUMBER will be equivalent to <i> CUSTOMER.

Remember that, as previously,

<i> NUMBER pushes the value of the ith STAR or CUSTOMER on the stack.

and <m> TO <i> NUMBER will store <m> into the ith STAR or CUSTOMER .

At any time, CUSTOMER can be replaced by STAR , (and vice versa) by

```
    ' STAR MATCH NUMBER
```

In this way, MEMORY is defined by

```
      VARRAY  MEMORY
    Ø MATCH   MEMORY
```

;S PB

```
SCR # 150
 ( Example of the creation of TO )
 HERE 0 , CONSTANT ZVAR   1 ZVAR SET TO
 : VAR   CONSTANT ;CODE INB,  ZVAR LDA,
        AO IF,  B I) LDA,  PUSH,
              ELSE,  CLA,  ZVAR STA,  S) LDA,  B I) STA, POP,
              THEN,
 : DVAR   CONSTANT  ,  ;CODE  INB,  ZVAR LDA,
        AO IF,  B LDA,  DSP,  A I) DLD,  S) STB,  PUSH,
              ELSE,  CLA,  ZVAR STA,  ..T STB,  S) DLD,
                       ..T I) DST,  POP.,
              THEN,
 : ARRAY   0  CONSTANT  DP  +!  ;CODE  INB,  S) ADB,  ZVAR LDA,
        AO IF,  B I) LDA,  PUT,
              ELSE,  CLA,  ZVAR STA,
                       S1) LDA,  B I) STA,  POP.,
              THEN,

SCR # 151
 : 2ARRAY   0  CONSTANT  DUP  +  1+  DP  +!
       ;CODE  INB,  S) ADB,  S) ADB,  ZVAR LDA,
         AO IF,  B I) DLD,  S) STB,  PUSH,
               ELSE,  CLA,  ZVAR STA,  ..T STB,
                        ISP,  S) DLD,  ..T I) DST,  POP.,
               THEN,
 : VARRAY  0 CONSTANT  ;CODE  INB,  B I) LDB,  S) ADB,  ZVAR LDA,
         AO IF,  B I) LDA,  PUT,
               ELSE,  CLA,  ZVAR STA,  S1) LDA,  B I) STA,  POP,
               THEN,
 : DVARRAY  0 CONSTANT ;CODE INB, B I) LDB, S) ADB, S) ADB,
         ZVAR LDA,  AO IF, B I) DLD,  S) STB,  PUSH,
               ELSE,  CLA,  ZVAR STA,  ..T STB,
                      ISP,  S) DLD,  ..T I) DST, POP.,
               THEN,
 FORTH   IMP  '  : MATCH  '  !  ;    IMP  '
```

Paul's' example of the use of TO will be presented in the next issue of Forth Dimensions. It utilizes Knuths' example for the calculation of the dates of Easter.

FORTH IMPLEMENTATION PROJECT

In June of 1978, the Forth Interest Group held its' first public meeting. With only mimimal publicity, we had in excess of 40 people attend. We had intended to offer educational assistance in using Forth. However, we found everyone was enthusiastic to learn Forth, but only five had access to running systems.

FIG then surveyed for vendor availability of the language. We found there were numerous mini-computer versions at educational and research institutions, all directly decended from Mr. Moore's word at NRAO. Of course, Forth, Inc offers numerous commercial systems.

None of these systems were available for personal computing. It appeared unlikely that this need would be met in the forseeable future. Our conclusion was that a suitable model should be created, and transported to individual micro-computers. Thus was born the Forth Implementation Team (FIT).

This team was proposed as a three tier structure. The first tier had several experienced Forth systems programmers, who would provide the model and guide the implementation effort. The next tier was the most critical. It was composed of systems level programmers, not necessarily having a background in Forth. They were to transport the common language model to their own computers by generating an assembly language listing that followed the model. Their results would be passed to the distributors that form the third tier. These distributors would customize for specific personal computer brands. Finally, the users could have access to both source and object code for maintenance.

Space doesn't permit inclusion of the FIT project, itself. The project was detailed as one of the six Forth conference papers at the Fourth West Coast Computer Faire, May 1979 in San Francisco. [1] The result is that FIG now offers the Installation Manual with glossary and Forth model ($10.00) and assembly language listings for numerous computers

($10.00 @) Included are: 8080, PDP-11, PACE, 9900, 6800, and soon 6502, and Z-80.

Note that FIG offers these listings which still have to be edited into machine readable form, customized, and assembled for specific installations. We hope that local teams share the effort and then distribute for others.

Reports of installations are beginning to come in from the USA and Europe. We sincerely hope this work will give a benchmark of quality and uniformity that will raise the expectation of all users.

FIG would like to thank the following members of the Implementation Team who have devoted a major part of nine months' spare time to this effort.

Dave Boulton	Instructor
John Cassady	8080
Gary Feierbach	Comp. Aut.
Bernard Greening	Z-80
Kim Harris	Librarian
John James	PDP-11
Dave Kilbridge	PACE
Dave Lion	6800
Mike O'Malley	9900
Bill Ragsdale	Instructor
Bob Smith	6800
LaFarr Stuart	6800

[1] Ragsdale, William F.
"Forth Implementation, A Team Approach"
The Best of the Computer Faires, Vol IV
from: Computer Faire ($14.78, USA)
333 Swett Road, Woodside, CA 94062

FORTH INTERNATIONAL STANDARDS TEAM

For several years the Forth Users Group (Europe) has sponsored a team working toward a standards publication for Forth. The 1977 meeting (Utrecht) produced a working document FORTH-77. Attendees included European educational institutions and Forth, Inc.

In October, 1978, an expanded group met at Catalina Island (Calif.). Attendees included Forth Users Group (Neiuwenhuigzen, Bartholdi), Forth, Inc. (Moore, Rather, Sanderson), FIG (James, Boulton, Ragsdale, Harris), Kitt Peak (Miedaner, Goad, Scott), U of Rochester (Forsley), SLAC (Stoddard), and Safeguard Ind. (Vurpillat).

The document resulting from this four day meeting has been released as FORTH-78. This document is becoming a good reference guide in evaluating the consistency and completeness of particular Forth systems. It is available from FIST to participating sponsors. (See below.)

A major benefit of the team meeting was the development of close communications between major users. For example, FIG is learning from the multi-tasking of U of R, Kitt Peak and Utrecht are running fig-FORTH, and we have adopted the security package pioneered in Europe. None of these events would have been likely without the contacts begun at Catalina.

The Team has announced the next Standards Meeting for October 14 thru 18, 1979, again at Catalina. The team agreed on an orginizational budget of $1000.00, to be met by $30. contribuitions by sponsors (individuals and companies).

These funds will be used solely to defray organizing costs of the annual meeting and distribution of the working documents to participants. Those considering participating should become Team Sponsors by remiting as given below. Sponsors will receive the just released FORTH-78, and all Team mailings.

Please remit to FIST, %Carolyn Rosenberg, Forth, Inc. 815 Manhattan Ave., Manhattan Beach, CA 90266.

PASCAL COMPUTING SERVICES, INC.

1979 February 22

Editor
FORTH DIMENSIONS
P.O. Box 1105
San Carlos, CA 94070

Dear Sir:

I was somewhat disconcerted when I read the article by Mr. David J. Sirag, "DTC Versus ITC for FORTH on the PDP-11", FORTH Dimensions, Volume 1, No. 3. The author has, I believe, misunderstood the intent of the article by Mr. Dewar.

In Mr. Dewar's article, the definitions of direct threaded code (DTC) and indirect threaded code (ITC) are

> "DTC involves the generation of code (my emphasis) consisting of a linear list of addresses of routines to be executed."
>
> "ITC..." (involves the generation of code consisting) "...of a linear list of addresses of words which contain addresses of routines to be executed."

As applied to the FORTH type of heirarchal structure (Heirarchal indirect threaded code?), I would extend Mr. Dewar's definition to be

> "ITC involves the generation of code consisting of a linear list of addresses of words which contain addresses of routines to be executed. These routines may themselves be ITC structures."

However, Mr. Sirag based his conclusions on the following loose definition:

> "The distinction between DTC and ITC as applied to FORTH is that in DTC executable machine code is expected as the first word after the definition name; while, in ITC the address of the machine code is expected."

Obviously the two men are not referring to the same things. Mr. Dewar is referring to the list of addresses which define the FORTH word, while Mr. Sirag is referring to the implmentation of the FORTH interpreter. If indeed Mr. Sirag's statement were true (which it is not) that their "analysis contradicts the findings of Dewar", then they should have implemented a DTC language rather than the ITC language of FORTH! Indeed, a careful examination of what is actually occuring in LABFORTH reveals that their techniques are logically identical to Dewar's ITC. They have simply, through clever programming, taken advantage of a particular instruction set and architecture. It is beyond the scope of this letter to prove this equivalence, or to support the FIG desire to have a common implementation structure for all versions of FIG FORTH.

Please note that I am <u>not</u> quibling over semantics with Mr. Sirag. All definitions are arbituary. (However, the value of a definition lies in its consistency, precision, and useability. I find Mr. Sirag's definition of DTC and ITC to be inconsistant with the environment in which he operates, FORTH, and thus quite useless.) My intent is two fold: (1) I am a self appointed defender of the excellent work of Mr. Dewar, and (2) I want to correct any misconceptions concerning this issue for readers of this newsletter who did not have access to Dewar's (better) definition of DTC and ITC.

Sincerely,

Jon F. Spencer

Jon F. Spencer
President
Pascal Computing Services, Inc.
14011 Ventura Blvd., Suite 201E
Sherman Oaks, CA 91403

poly-FORTH BY FORTH, INC

Because of its speed and economy, FORTH Inc.'s FORTH language has been favored by many mini and microcomputer designers. Now comes polyFORTH, which combines the best features incorporated in the mini and micro versions.

PolyFORTH is a multilevel language with the essential functions (e.g., basic arithmetic and logic operators) in a 512-byte nucleus, and user-defined "words" (comparable to macros) in the outermost layer. What's more, the standard package fits in 4 kbytes of PROM, with an additional 2 k for the assembler and text editor (which aren't needed to run a program once it's developed).

The new operating system goes beyond previous FORTH versions by being able to handle multitasking and many terminals (limited only by the hardware), and by including a buffer handler that supports RAM or mass storage. Other improvements over previous versions include faster dictionary search, all 16-bit arithmetic and a simpler target compiler.

The target compiler can be used to develop a program of a micro-computer development system. The compiler code can either be executed directly, or be compressed and burned into ROM. Scientific routines, a data-base management system and applications software are available options.

The 8080 and 9900 are polyFORTH's first targets, but versions for the 8086, LSI-11, Series-1 and Honeywell Level 6 are scheduled by the Manhattan Beach, CA company for release later this year.

Contact Steven Hicks at FORTH, Inc., 815 Manhattan Avenue, Manhattan Beach, CA 90266, (213) 372-8493.

January 23, 1979

Dear FIG:

Having just received your issue No. 3, it seems to me that the "DTC" method used by Sirag is still "indirect threaded code" in the terminology used by Dewar, and should be distinguished from other implementation methods as being "executed" rather than "interpreted." The use of actual machine code in place of the code address is the "executed" aspect, but only in the case of the Low Level Definition (Code) does the use of machine code reduce the level of indirection in the threading to that of direct threaded code. In the Storage Definition in Sirag's diagram, there is still a subroutine call in the dictionary entry containing data (constant, variable type entries). Direct threaded code would require this subroutine call to be moved from the individual data-Word to the code string referencing that word. Whether that subroutine call is executed by an actual machine language JUMP SUBROUTINE instruction or by an interpreter routine is another matter. Now, in the case of the Code type dictionary entry, use of executed machine code tends to also remove a level of indirection because only one jump is needed, there being only one address, that of the code routine, involved; this coincidence unfortunately confuses the two consequences, even though they are separate.

The concept of Threaded Code seems inherently fuzzy, because any high-level instructions compiled by a translator into a series of subroutine calls has the same form as threaded code, so it looks like almost any compiler is going to use a certain amount of threaded code. Some operations, however, like adding two numbers, take so little machine code that many compilers would expand them completely in-line instead of threading them in a subroutine, and it is at this low level of primitives that the concept has meaning. Even when a subroutine call saves nothing compared to a full in-line expansion, going to <u>two</u> subroutine calls, i.e. <u>indirect</u> threaded code, many save space by <u>reducing</u> the amount of code to identify the <u>types</u> of the operands being processed, and in some cases it may also save time. On a very large computer perhaps all the variable types involved are an inherent part of the machine instruction set, and the memory may be tagged to identify type there as well, and both the questions of executed/interpreted and indirect/direct would not be relevant. If one had a machine which executed FORTH primitives as its machine language, you would still have indirect threaded code, but completely executed instead of interpreted. The PDP-11 seems to fall in between.

None of the above considerations, however, contradict Mr. Sirag's main conclusion that how FORTH is implemented should not be part of its definition.

One final note: in the DTC, ITC comparison for storage definitions, DTC was shown having a larger overhead than ITC even though the VAR routine appears shorter in the DTC case. Even though the space overhead is greater, I wonder if he has overstated the DTC overhead time. He seems to show a single jump-subroutine taking longer than an interpreted jump.

Sincerely yours,

George B. Lyons
280 Henderson Street
Jersey City, New Jersey 07302

November, 1978

Dear FIG:

I was very excited to find something from you in my mail today, but then I was disappointed to discover that it was a copy of the journal article which introduced me to FORTH and your existence, which I already have.

On October second, I sent you a check and asked for everything else offered on the subscription form (FORTH DIMENSIONS, Volume 1, number 1, p. 22.), i.e., newsletter sub., glossary, and FORTH-65. And I've been anxiously awaiting the receipt of any or all of these.

Of course I realize you're all volunteers, and I'm not angry...but I really would like to get that stuff. I am, like many others, I imagine, anxious to get a version of FORTH up on my system. I've managed to dig up most of the references listed in the article (still waiting for DECUS) except for the last one: is it in print?

I also have the documentation for the Digital Group's CONERS and Programma's 6502FORTH (for the Apple, which I don't have). Both of these programs are outer interpreters written in assembly language, and contain no inner interpreter. Interesting, and they look like FORTH, but that's not really what I want to do...

Rather than invest another 30 cents in postage, I'm enclosing a check for $2.00 for the reprint--I know these things aren't free, but pleaes send the other items soon, okay? Thanks.

Sincerely,

Dan del Vecchio
Ann Arbor, MI

Editor --

Apologies to Mr. del Vecchio. We mishandled his entire request, and yet he encloses an extra $2.00! Our mail processing is now current. We will complete the full six issues of FORTH DIMENSIONS.

```
SCR # 3

                 GLOSSARY DOCUMENTATICN - D.W.BORDEN
     AFTER READING W.F.RAGSDALE'S ARTICLE ON THE "HELP" COMMAND
IN FORTH DIMENSIONS NO.2 AND SOME PROPAGANDA FROM FORTH INC., I
WROTE A SHORT PROGRAM WHICH PRINTED OUT EACH FORTH WORD AS IT
IS DEFINED, THE ADDRESS OF THE LENGTH BYTE AND THE ADDRESS OF
THE PARM BYTE. AFTER EACH ENTRY, I PRINTED ELLIPSES TO ALLOW
A HANDWRITTEN ENTRY OF WHAT EACH COMMAND IS SUPPOSEDLY DOING.
THE ADDRESS OF THE PARM BYTE IS USEFUL SINCE THAT ADDRESS
APPEARS IN HIGH LEVEL COLON DEFINITIONS "OBJECT" CODE. THUS, IF
YOU HAVE NOT WRITTEN YOUR OWN FORTH SYSTEM, YOU CAN HAND
DISASSEMBLE ( DISFORTH MIGHT BE MORE APPROPRIATE ) EACH COMMAND
AND SEE WHAT IT IS DOING.
   OF COURSE A DISFORTH PROGRAM COULD BE WRITTEN AND I HAVE DONE
SO, BUT I AM NOT HAPPY WITH ITS OUTPUT FORMAT YET. I ALSO HAVE
WRITTEN A TRACE WHICH PUTS A TRAP IN "NEXT" AND LISTS EACH FORTH
COMMAND EXECUTED. VERY USEFUL FOR DEBUGGING.     ;S  01/31/79
SCR # 2

( GLOSSARY OF FORTH WORDS WITH HEAD AND PARM ADDRESSES )
0 VARIABLE CMD  ( TEMPORARY VARIABLE TO HOLD COMMAND )
: TOPOFPAGE CR CR [ CMD    HEAD  PARM ] CR ;
: UNDERLINE [ ........................................ ] ;
: GLOSSARY TOPOFPAGE CURRENT @ @ CMD ! ( GET TOP CMD ADDRESS )
BEGIN
CMD @ IF CMD @ C@ DUP 80 AND - ( GET COMMAND LENGTH )
DECIMAL . HEX 4 1 DO ( PRINT COMMAND LENGTH )
CMD @ I + ( INDEX FETCHED IS 1-2-3 )
C@ ECHO ( PRINT COMMAND LETTER )
LOOP SPACE
CMD @ .H SPACE ( PRINT COMMAND HEAD ADDRESS )
CMD @ 6 + .H UNDERLINE CR CR ( PRINT CMD PARM ADDRESS )
CMD @ 4 + @ CMD ! ( PICK UP LINK WITH NEXT CMD ADDRESS )
ELSE QUIT THEN
AGAIN ;                 ;S       1/30/79       DWB
```

example	fig-FORTH
[	."
]	"
ECHO	EMIT
.H	H. (hex output)

```
GLOSSARY

CMD    HEAD  PARM
8 GLO  1C2A  1C30........................................

9 UND  1BEE  1BF4........................................

9 TOP  1BC9  1BCF........................................

3 CMD  1BBD  1BC3........................................

9 ?TE  1BA1  1BA7........................................
```

Dear FIG:

I have been programming for sixteen years, but I only discovered FORTH about two weeks ago! It was a clear case of love at first sight, but I've encountered a really sticky problem already. I have seven different programs, each runs on my Apple II computer, and each claims the name "FORTH". Aside from that the only thing they have in common are the three verbs, ":", ";", and "@". Beyond that, everything is different.

My question is: Is there such a thing as a "standard" FORTH, and if so, where do I find out more about it? The only documentation I have rounded up so far is one borrowed copy of FORTH DIMENSIONS, and two pitifully incomplete "user's guides" supplied with two of the versions FORTH I've bought. HELP!

Sincerely,

Gary J. Shannon
14115 Hubbard Street
Sylmar, CA 91342

Editor --

Mr. Shannon addresses a major problem! FORTH uniformity is a major problem. Current standards (FORTH-77 and FORTH-78) exist but cover only areas of mutual user concensus. There are remaining areas where definition and/or refinement are needed. The FIG Installation MANUAL has a model of the language offered for public comment toward uniformity. We also pin-point areas of deviation from FORTH-78 in our publications.

Dear FIG:

I have an 8k PET and after reading Dr. Dobb's number 28 I called Programma Consultants to purchase FORTH. Two weeks later I received it!

I have been working with PET FORTH now for a couple of weeks and am still fascinated with FORTH although I have some problems with Programma's implementation.

Please sign me up.

Regards,

Chris Torkildson
St. Paul, MN

December, 1978

Dear FIG:

The FORTH-65 implementation listing you sent is great!!! I'm beginning to understand the power and beauty of this language/system, with the aid of it, a Caltech (Space Radiation Lab) FORTH manual, and James' article in Dr. Dobbs...but there are still some items I can't figure out. (One stupid question: What is PROTECT, used in screens 36 and 45?)

Also, have you reviewed Programma International's PET FORTH yet? They told me that a new version (1.1) would be out in Nov./Dec., but I'm waiting before buying...would like your opinions/recommendations.

Best,

Mark Zimmerman
Caltech 130-33
Pasadena, CA 91125

Editor--

Programma International's version is not recommended by FIG. It has a non-standard header (no word length indication) and is pure machine code. The inner interpreter NEXT is missing as are the critical definitions ;CODE, <BUILDS, DOES>, and BLOCK. PROTECT traps execution of compiling words outside of colon-definitions. FIG now uses ?COMP for this purpose.

April, 1979

Dear FIG:

I now have Programma's 6800 FORTH and I probably shouldn't complain, however it does have some differences with the DECUS FORTH (CALTECH FORTH). I think my major complaint is with the coding; any conditional branching is almost impossible without a previously assembled listing to obtain address displacements. My MIKADOS/OS/assembler/disassembler) is interactive, single-pass, and fast. I haven't figured out yet how to do any editing other than backspace and line deletion, other than redo the program. Also, I have the Felix USP and recursive programs just come naturally; same with textual programming - somewhat difficult with FORTH. Can recursive work be done in FORTH? I don't know as I can't find out enough from any manual (the only one with Programma's is "interim") to answer my questions.

My "down" outlook may surprise other Figgers, however I am (obviously) not a computer scientist but feel that after the Dobb's puffery by John James and a couple of letters to the editor, the simplicity and all-encompassment have been vastly overrated.

Sincerely,

Neal Chapion
Space #27
602 Copper Basin Road
Prescott, AZ 86301

Editor--
The above letters illustrate the negative comment we receive about Programma Internationals' FORTH.