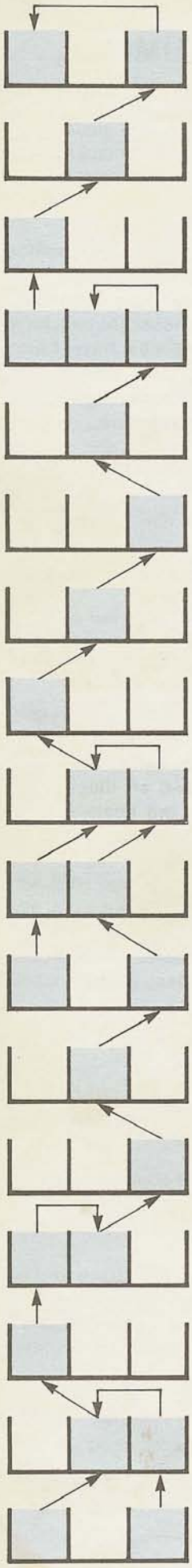


FORTH DIMENSIONS

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

Volume III
Number 1
Price $2.00

INSIDE

FORTH DIMENSIONS

Published by Forth Interest Group

Volume III No. 1 May/June 1981

Publisher Roy C. Martens
Guest Editor C. J. Street

Subscription to FORTH DIMENSIONS is free with membership in the Forth Interest Group at $12.00 per year ($24.00 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

The last edition of FORTH DIMENSIONS was the beginning of many changes in editorial policy and format. All these changes are designed to make FORTH DIMENSIONS a practical and useful communications tool.

This practical approach continues. In this edition you will find a number of utility programs that will make the task of implementing practical applications in FORTH easier and faster. All of these utilities have been contributed by FIG members who have found them to be valuable tools. The editorial staff thanks these contributors and would like to encourage all FIG members to share their ideas and experience.

If you have a programming idea or tool that you have found useful, please send it to:

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

YOU DON'T HAVE TO BE A WRITER—the editorial staff will provide whatever copywriting is necessary to make your ideas publishable.

On the aesthetic side, you will find this edition contains several photographs and art illustrations. This is a FORTH DIMENSIONS' first and you can expect to see more in the future. Photographs and art illustrations will be chosen and published on the basis of their educational and human interest value.

This issue also introduces the character HEX who will be FORTH DIMENSIONS' official comic strip. You will find the HEX comic strip in future editions of FORTH DIMENSIONS. HEX's adventures will be a combination of humor and education. Ideas for HEX comic strips are welcome.

C. J. Street
Editor

PUBLISHER'S COLUMN

Renewals and new members are coming in at a fast pace. We expect to climb to 3,000 members in the next few months and to 5,000 within a year.

Both the Computer Faire in San Francisco and the Computer Festival in Trenton, New Jersey were huge successes. We'll be in both again next year. (I'd like to know of any other shows where you think FIG should exhibit.)

Good material is coming in for FORTH DIMENSIONS. Keep it coming and send in your comments.

Roy Martens

LETTERS

Dear Fig:

My company is developing business systems using FORTH and we would be interested in communicating with local FIGGIES as well as offering our input to FST, FORML, FIG and other applicable "F" acronyms since it is obviously in our interest to promote the spread and acceptance of FORTH. We're also confirmed FORTH fanatics.

David B. Moens
BUSINESS SYSTEMS SOFTWARE, INC.
27 East Kings Highway
Haddonfield, NJ 08033
(609) 429-0229

You are our kind of fanatic and we're happy to put the word out for you Dave! -- ed.

Dear Fig:

Re: "Born-again programmer" and "Born-again FIGGER" in FORTH DIMENSIONS II/5.

My interest in FORTH as a programming language does not envolve becoming mired down in the morass of a religion. It would be best to stay on rational grounds in the development of FORTH and leave religion to those who are unable to think without faith.

I will not take part in a religious group. Therefore I will not be renewing my membership.

Larry R. Shultis
P.O. Box 218
Fontana, WI 53125

Just goes to show you that there is more than one type of fanatic! Keep the faith, Larry, (OOPS, sorry about that! I meant: Don't worry, Larry,) FIG is not envolving (your spelling) into a religious group. -- ed.

Dear Fig:

Thank you for the prompt and efficient service I have received. I realize that you can't have much time left to look after the rest of the world, but without your interest it may never have reached these shores. Spread the good WORD.

J. Huttley
UNIVERSITY OF AUCKLAND
19 Duncan Avenue
Auckland 8,
New Zealand

You are very welcome! -- ed.

Editor's note:

At the WEST COAST COMPUTER FAIRE in California two versions of a FORTH bumper strip were circulated:

```
?FORTH IF HONK THEN
```

or alternately

```
: LOVE-FORTH
    IF HONK THEN  ;
```

Just thought you might like to know. -- ed.

Dear Fig:

TGIF is very interested in swapping listings with other Fig-FORTH groups. Our current listings are 2 Decompilers; Full screen editor; CASE statements for 8080, Z80 and 6502; 6502 Assembler.

TGIF
FORTH INTEREST GROUP -- TULSA
Box 1133
Tulsa, OK 74103

How about sending them in to FORTH DIMENSIONS? -- ed.

Dear Fig:

I was lucky enough to attend one day of the recent West Coast Computer Faire and to meet some of the mentors of FIG. I had numerous questions and enjoyed talking to Bill Ragsdale and others about them. (By the way, for those of you who couldn't make it to the Faire, the FIG booth was one of the most crowded. People were standing there like no where else even as the 5:00 closing approached. We all owe a thanks to the dedicated folks for their time and effort in this endeavor, promoting and spreading the word of FORTH and FIG.)

One of my questions to Bill was "How can we remote members contribute to FIG" in ways other than articles for FORTH DIMENSIONS. I got a number of project ideas, for one of which I need the help of the whole membership. So PLEASE: NOW HERE THIS!

I propose to assemble a book of utility program packages for publication by FIG. I have a list of functions which I think should be included. This covers such things as editors (both the current FIG line oriented editor and a video screen type editor), string processing, data structures, extended math (double precision as an example), math functions (sin, log, etc.), matrix operations, and floating point routines. No doubt there are others to be considered and I solicit your suggestions.

The plan I propose to compile and publish such a document is as follows:

1. Members are asked to send their proposals for implementing utility packages to me at the above address (or through FIG). These proposals should consist of well documented (lots of comments) fig-FORTH source code accompanied by complete instructions for use, any known operating limitations, and a brief technical description or reference if appropriate. The programs should be as transportable as practicable; if system unique routines are necessary, enough information should be provided so as to allow them to be adapted to a different machine.

2. I will compile a draft of the proposed publication and submit it to a technical review committee for review and appropriate testing. This committee of FIG members (I am looking for volunteers) will scrutinize the proposals (and alternatives if appropriate), test them on a running FORTH system, and make recommendations.

3. I will then compile the final version and submit it to FIG for publication.

I have set some timelines for compiling this compendium such that it can be published by next year's Computer Faire. Material should be sent in no later then 1 September 1981 (please send them early and give me a chance to get ahead). I will distribute the draft for review and testing by 15 October 1981. Finally I will begin compiling the final version by 1 January 1982 and have it ready for publication by 1 February 1982.

This may seem like a long time abuilding, but I want to provide ample opportunity for the contributors to develop their ideas fully and conduct a fair amount of testing themselves before submission, I also want to provide a good review by the committee to provide the highest quality document for FIG. It will be worth it in the long run. Your contributions will be sincerely appreciated, and though the publication, as are all of FIG's, will be in the public domain, credit will be given to the authors and contributors. So if nothing else, this is a chance to get your name in print, in an international publication.

Sincerely,
George O. Young III
617 Mark NE
Albuquerque, NM 87123

This is a great project. Our goals continue to be to decentralize FIG activities, and George's project of a published "Goodies Package" will be helpful to all. Contributors should send a brief description to George and then prepare the full document. This will allow co-ordination of similar material. -- ed.

Spreading the fig-FORTH at the West Coast Computer Faire, April, 1981.
Top: (l-r) Michel Mannoni (FORTH vendor), Dave Boulton and Martin Schaaf (Answering the question: What's FORTH? 10,000 times)
Bottom: (l-r) Roy Martens (FD Publisher), Anne & Bill Ragsdale (FIG prime movers).
Order your T-shirt, like Bill's!

ANNOUNCEMENTS

FORML CONFERENCE CALL FOR PAPERS

Papers are requested for the three day technical workshop to be held next November 25th to 27th at the Asilomar Conference Grounds in Pacific Grove, California (Monterey Peninsula).

Although registration for this conference is not yet open, attendance will be limited to 60 persons. Authors will be accepted before listeners, so if you want to attend, the only sure way is to write a paper. Please note: abstracts or proposals for papers or discussions must be submitted no later than September 1, 1981 for inclusion in the conference and its proceedings; completed papers by September 15, 1981.

The purpose of this workshop is to discuss advanced technical topics related to FORTH implementation, language and its applications. Potential authors should write for an author's packet for detailed instructions. Send all correspondence regarding the conference or papers to:

FORML CONFERENCE
P.O. Box 51351
Palo Alto, CA 94303

FORTH WORKSHOPS

Beginners and advanced workshops in FORTH are being jointly sponsored by the College of Notre Dame and Inner Access Corporation both of Belmont, CA.

Beginners workshops start June 23 and advanced workshops start July 14. Classes meet every Tuesday and Thursday evening from 7:00 to 9:00 P.M. Registration is $135 for 3 weeks (12 hours).

For more information and registration contact:

College of Notre Dame (415) 593-1601

CONTINUING DIALOG ON FORTH-79 STANDARD

Dear Bill:

We recently obtained a copy of the FORTH-70 Standard from FIG and are attempting to align our version of FORTH with it. The document is generally well done and in most cases clearly and concisely expressed. However, there are about a half dozen or so definitions that seem to us somewhat ambiguous.

I am writing to you in the hopes that you can clarify the word definitions in question; or, that you can refer us to someone who can. I am also interested in knowing whether the FIG model has been aligned yet, if not, when it will be.

My list of questions is enclosed and I would appreciate anything you can do to assist us in their resolution.

Cordially,

Robert D. Villwock
MICROSYSTEMS, INC.
2500 E. Foothill Blvd., #102
Pasadena, CA 91107

OPEN QUESTIONS
FORTH-79 Standard

1. For the words / and */ does the terminology "rounded toward zero" essentially mean truncated? If not, precisely what does it mean?

2. The word SGIN is now apparently defined to be used "outside" of the <#, #> operators. What is the precise definition of where the minus sign character is to be stored? Why was this word changed from its former function between <# and #> ?

3. The word ':' is defined as a non-precedence word. Is this a typo or is it intentional? If intentional, could you explain the rationale? It seems that the number of occasions for which "colon" needs to be compiled are few and could easily be handled by using the [COMPILE] operator. ON the other hand, syntax errors and typos often result in mistaken attempts to compile ':' which, when it's an immediate word, can be flagged by the compiler.

4. The word CODE is defined as using the form:

   CODE <name> ...END-CODE

   However, the word ;CODE says nothing about the corresponding form. Our version of FORTH rquires that code level action routines defined by ;CODE also be terminated by the word END-CODE. Is this compatible with FORTH-79?

5. The words FIND, ',' ':', etc., as defined in the Standard, indicated a search of CONTEXT and FORTH only. Is it considered an incompatibility if the CURRENT vocabulary is also searched (if different)? The definition of VOCABULARY is not clear regarding the possibility of "sub-vocabularies" such as ABC chained to XYZ chained to FORTH. If this is allowed, and, ABC is the CONTEXT vocabulary, is not ABC, XYZ, and FORTH searched?

6. What is the mnemonic significance of the C words such as C!, CMOVE, etc.? Surely it doesn't stand for "cell," does it? The term "cell" is defined on page 3 of the Standard to be a 16-bit memory location. The word MOVE is defined on page 26 to transfer 16-bit words ("cells"), while the word CMOVE on page 20 is defined to move bytes (not "cells"). If the C does stand for "cell" what is the rationale? Why was the former standard's B (for byte) replaced by the mysterious C?

7. I note that in the reference section of the Standard, the word DPL which formerly used to handle both input and output "point" situations now strongly emphasizes that input conversations should not affect its

value. What is the reason for this restriction? How likely is it that this may become part of the Standard?

8. The definition for CREATE is not clear. Does the second sentence "When <name> is subsequently executed, the address of the first byte of <name> 's parameter field is left on the stack" mean that the word CREATE alone is to function this way or only when followed by ;CODE or DOES> ? In other words, is it intended that CREATE work as in the FIG model or has its definition changed? Taken literally, FORTH-79 says that CREATE will generate an unsmudged header with the CFA pointing to the run time procedure for variables. Is this what is intended?

COMMENTARY FROM THE FORTH DOCTOR

1. Some computers apparently (by Standard Team comment) round quotients and remainders to smaller magnitude (more negative). Trucation of negative quotients would do this. If a correct representation is not possible, the result should be nearer zero. Dave Boulton is more knowledgeable on this point.

2. " Sign is to be used within <# and #>. The user chooses where to store the sign. Notice that no word generates the saving of the sign. In fig-FORTH the only difference is the ROT would be explicitly done just before SIGN, rather than in SIGN.

3. FIG and the Europeans make : an immediate word for error control. Other users, and FORTH, Inc. reject this level of error control--too bad! We need a technical paper presenting the trade-offs (code needed and compilation slowdown). Conversation at a team meeting is insufficient to change opinions developed over ten years.

4. These topics were barely touched on by the Team as CODE definitions are not portable. ;CODE probably should terminate in END-CODE. This is an unresolved area.

5. The standard wording was painstakingly done regarding vocabularies. This is the most divergent topic among users. all known methods can comply with the Standard, but it does less than all systems. The rationale is that you build CURRENT but you execute only from CONTEXT (and FORTH). No chaining is recognized, beyond context leading to FORTH. This may be physical links or logical (within FIND). Again, position papers are essential to get a common, more advanced, construct.

6. Charles Moore has used C for ten years as a character (byte) prefix. Ignore (if you can) that a character is defined as 7 bits in the Standard. This was a hotly disputed point with FIG and the Europeans for "B"yte and FORTH, Inc. and a couple of others for "C". Kitt Peak was adamant before the meeting for "B" and other uniformity improvements. Their representatives made no defense of the issue. Historical precedence wins this one.

7. Reference Section is just leftovers. Only one vote of any team member was sufficient to maintain a Reference word on the list. The Standard attempts to minimize system variables. Increased usage of special variables is unlikely. Things like DPS are delegated to applications.

8. The definition of CREATE is quite clear. You have stated it and then correctly paraphrased it. Other defining words may be used before DOES> which help build a parameter field. DOES> rewrites the code field to its own code.

```
: CPU CONSTANT DOES>  ;
```

is equivalent to

```
: CPU CREATE , DOES>  ;
```

TECHNICAL NOTES, BUGS & FIXES

Dear Fig:

I have recently brought up FORTH on a 6800 system and find it to be a very easy and powerful system for microcomputers.

I have a mini-computer with a cross-assembler on it which I used to assemble the source after keying it in. Naturally, as soon as I got it working I wanted to change it. I feel that the EXPECT routine and backspace handling could be improved significantly by incorporation of the enclosed recommendations.

I also experimented with the GLOSSARY routine submitted by D.W. Borden in FORTH DIMENSIONS, Volume 1, No. 4. I modified it to handle the variable length name field and changed the format slightly.

Keep up the good work.

Toby L. Kraft
San Diego

```
   EXPECT
001 ( EXPECT :: MODIFIED FOR USER DEFINED BACKSPACE CHAR )
002 ( 26FEB81   TOBY L. KRAFT )
003 : EXPECT
004 OVER + OVER          ( ADD COUNT TO ADDRESS FOR LOOP LIMIT )
005 DO
006   KEY DUP            ( GET CHAR AND SAVE COPY )
007   0E +ORIGIN @ =     ( GET SYSTEM BACKSPACE AND CHECK FOR IT )
008   IF
009     DROP DUP 1 =       ( LOSE CHAR , CHECK BUFFER BEGIN )
010     DUP R> 2 - + >R    ( ADJUST BUFFER POINTER APPROPRIATELY )
011     IF 07 ELSE BSTOF C@ ENDIF  ( BELL @ BEGINNING , BS OTHERWISE )
012   ELSE
013     DUP 0D =         ( CHECK FOR CARRIAGE RETURN )
014     IF
015       LEAVE DROP BL 0    ( PREPARE TO LEAVE )
016     ELSE DUP ENDIF
017     I C! 0 I 1+ !      ( STORE CHARACTER IN BUFFER )
018   ENDIF
019   EMIT                     ( ECHO CHAR TO TERMINAL )
020 LOOP
021 DROP
022 ;
```

```
   GLOSSARY
001 ( GLOSSARY GENERATOR ROUTINES )
002 DECIMAL
003 0 VARIABLE CMD
004 : TOF CR CR 32 SPACES ." GLOSSARY" CR CR ( GENERATE PAGE HEADING )
005   ." LEN WORD" 15 SPACES ." NFA    PFA" CR CR ;
006 HEX
007 : GLOSSARY TOF CONTEXT @ @ CMD !
008 BEGIN CMD @ IF
009   CMD @ C@ 3F AND
010   DECIMAL DUP 3 .R SPACE
011   F SWAP - CMD @ ID. SPACES
012   CMD @ HERE 6 .R SPACE
013   CMD @ 1 TRAVERSE DUP
014   3 + 6 .R SPACE CR
015   1+ @ CMD !
016   ?TERMINAL IF QUIT ENDIF
017   ELSE QUIT THEN AGAIN ;
```

Modifications to the fig-FORTH boot-up literals:

1. Backspace Character
   Character to emit in response to a backspace entry. X'08' (control-H) is character FORTH responds to for backspace function. Character to emit is terminal dependent and should be defined in the user table.

   This also allows use of a printable character (e.g. C'\') to emit for backspace for use on printing terminals.

2. Form Feed Character
   Character to emit to cause terminal (or printer) to advance to top of form. This is also device dependent and should be in user table.

3. Form Feed Delay
   Number of null characters to emit after issuing a form feed character. This is similar to CR/LF delay which is already provided.

Recommendation :

Add variable 'BSTOF' to user table.

X'BBFF' - two characters of data
- FF - form feed character (X'0C' initial value)
- BB - back space character (X'08' initial value)

Add word 'BSTOF' to vocabulary to access this variable in user table. (Similar to 'BASE')

Modify definition of current user variable 'DELAY' to include formfeed delay in upper byte.

Add word 'DELAY' to vocabulary to access this variable in user table.

Modify startup parameters and cold start accordingly.

Modify EXPECT to use user defined backspace character and to explicitly generate bell code (X'07'). Currently, EXPECT tests for the beginning of the buffer and subtracts the boolean flag result from X'08' to generate the character to emit in response to a backspace.

Toby L. Kraft
7822 Convoy Court
San Diego, CA 92111
(714) 268-3390

This really needs expansion and generality. How about terminals that need an "escape sequence" to clear screen, i.e. form feed? Toby, HEX should be used insteat of X'.--ed.

Dear Fig:

I wish to convey a concept which has greatly increased the clarity of my FORTH coding. It has to do with in-line documentation of the contents of the stack (comments within parathesis).

Unfortunately, none of the existing techniques (space, hyphens, brackets, or ordinal suffix) provide the brevity and clarity that one becomes accustomed to with FORTH. The technique which I have devised provides both. It revolves around the backslash character '\', which I refer to as 'under' and the double hyphen '--', which I refer to as 'leaves'. Using this terminology, the following comment:

```
( address\count -- )
```

is read "address under count leaves nothing," and

```
( N1\N2 -- N3 )
```

is read "Number1 under Number2 leaves Number3."

The 'under' symbol imparts a clear verbal and graphic representation of the ordering of the stack contents, and provides an elegant solution to a major problem encountered when transporting FORTH algorithms and source code.

Don Colburn
Creative Solutions, Inc.
4801 Randolph Road
Rockville, MD 20852

Dear Fig:

Some time ago I bought your Installation Manual and the 6502 Assembly Listing. I have been studying both for quite a while, and am also a charter member of the Potomac FORTH Interest Group (PFIG: Joel Shprentz and Paul VanDerEijk).

I have MMS FORTH (cassette) for the TRS80 up, and have just bought GEOTEC FLEX-FORTH for my KIM, although I don't have my 16K ram card installed in KIM yet. I do like FORTH!!! The PFIG has been fairly inactive for some time due to lack of a meeting place, but Joel Shprentz has been conducting some Intermediate FORTH classes ($30 for six lessons) which are ongoing, and very interesting - we are well into <BUILDS/DOES>, and will then go on to disking, etc. Ask Joel for details.

I'm still planning to bring up FORTH on the KIM from my own hand-assembled version, just to satisfy my own curiousity about what makes FORTH tick. I do think I'm finally beginning to understand how everything fits together.

In this vein, I have a few comments to pass on from an (advancing) novice FORTH enthusiast. The first two comments regard the above referenced Installation Manual and 6502 Assembly Listing. The last two are ideas of my own which I offer for what they are worth.

1. There is a disparity in the Installation Manual version of the 6502 memory map regarding the placement of the Disk buffer and User Area.

Indeed, there is disparity in the 6502 Assembly Listing between what is done near the front and what is actually implemented (per the installation Manual). The Installation Manual puts the Disk buffer at the top of RAM with the User area just below. Line 0051 of the assembly manual says User area is top 128 bytes, with disk buffer next (line 0052). CREATE assumes just the oppposite in both the Installation Manual and Assembly Listing. (Editor -- correct on all all points. The author was inconsistent.)

2. In screen 49 of the Installation Manual, I see no need whatsoever for a dedicated word such as ID. to move the word name to Pad and then type it out! The first 4 words are not needed, and neither are the words following " - " (PAD SWAP CMOVE PAD). Just a waste of time and space to bring the name to PAD and then type it out! (Editor -- this is not so. If you have WIDTH set to less than 31, ID. is required.)

3. I would suggest a word (Q) that might be inserted into any type of loop (DO/LOOP or BEGIN/AGAIN) to allow a timely exit when things go awry (as they do with Novices!). It's very simple - : Q ?TERMINAL IF QUIT ENDIF ; MMS FORTH has this embedded into the code of " : " , but I think that's overkill. But it sure is nice to undo errors put into loops. (Editor -- this is terrible style. LEAVE is the correct way for a controlled termination.)

4. This has specifically to do with the Jump Indirect of the 6502 as used in both the Installation Manual and the assembly listing. Having used the 6502 for better than 4 years, I have yet to use the JMP indirect after finding out about its shortcoming of wrapping around within a page if low byte of address is $FF. I pretend this opcode does not exist. (Editor -- CREATE on 6502 systems correctly places code field. Anymore comments should be directed to Chuck Peddle, designer of 6502.)

Keep up the good work.

Edward B. (Ted) Beach
5112 Williamsburg Blvd.
Arlington, VA 22207

CORRECTION ON SEARCH

by John James
(Vol. II #6)

When you are debugging or modifying a program, it is often important to search the whole program text, or a range of it, for a given string (e.g. an operation name). The 'SEARCH' operation given below does this.

To use 'SEARCH', you need to have the FIG editor running already. This is because 'SEARCH' uses some of the editor operations in its own definition. The 'SEARCH' source code fits easily into a single screen; it is so short because it uses the already-defined editing functions. Incidently, the FIG editor is documented and listed in the back of FIG's Installation Manual.

Use the editor to store the source code of 'SEARCH' onto a screen. Then when you need to search, load the screen. (Of course if you are using a proprietary version of FORTH, it may have an editor and search function built in and automatically available when needed. This article-ette is mainly for FORTH users whose systems are the ten-dollar type-it-in-yourself variety).

Here is an example of using 'SEARCH'. We are searching for the string 'COUNT' in screens 39-41; the source code of 'SEARCH' is on screen 40. The screen and line numbers are shown for each hit. Incidently, the search string may contain blanks. Just type the first screen number, the last screen number, 'SEARCH' followed by one blank and the target text string. Conclude the line with return. The routine will scan over the range of

screens doing a text match for the target string. All matches will be listed with the line number and screen number.

Happy SEARCHing!

```
39 41 SEARCH COUNT
 00 VARIABLE COUNT_ER                    2 40
    1 COUNT_ER +!  COUNTER @             4 40
    1 COUNTER +!  COUNT_ER @             4 40
    56 >IF   0 COUNT_ER !                5 40
    12 EMIT 01 TEXT  0 COUNT_ER!         8 40  OK
```

```
SCR # 40

 0 ( SEARCH, OVER RANGE OF SCREENS     WFR)
 1 DECIMAL
 2 00 VARIABLE COUNTER
 3 : BUMP    ( THE LINE NUMBER AND HANDLE PAGING)
 4    1 COUNTER +!   COUNTER @
 5    56 > IF   0 COUNTER !
 6       CR CR 15 MESSAGE 12 EMIT THEN ;
 7 : SEARCH  ( FROM, TO --- TARGET STRING )
 8    12 EMIT 01 TEXT   0 COUNTER !
 9    1+ SWAP  DO  FORTH  I  SCR  !
10    EDITOR TOP
11    BEGIN 1LINE IF 0 M SCR ? BUMP THEN
12      1023 R# @ < UNTIL
13    LOOP ;  CR  ." SEARCH IS LOADED  " ;S
14
15 TYPICAL USE TO LOCATE 'KEY-WORD':   21  44  SEARCH KEY-WORD
```

```
 8 : .S   ( prints stack contents, top last;
    stack unchanged )
 9    DEPTH IF   SP@ 2-  S0 @ 2-  DO   I ?
    -2 +LOOP
10         ELSE   ." Empty "   THEN   ;
11
12 ( which vocabulary is being referenced? )
13 : .VOC   (prints CONTEXT VOCABULARY name )
14    CONTEXT @  4 -   NFA ID.   ;
15
```

```
SCR # 82
 0 ( Tools: number printing   fig-FORTH 1.x )
 1
 2 : .BASE   ( — ) (prints current radix in
    decimal )
 3    BASE @ DUP   DECIMAL .  BASE !    ;
 4
 5 ( create base-specific stack-print operators )
 6 : BASED.   (<BUILDS:  newBase — )
    ( DOES>:  n — )
 7      BUILDS>,
 8    DOES>@  BASE @  SWAP BASE !   SWAP .
    BASE !   ;
 9
10 16 BASED. H.   ( print top-of-stack in hex )
11  8 BASED. O.   ( print in octal )
12  2 BASED. B.   ( print in binary )
13
14
15
OK
```

PROGRAMMING AIDS & UTILITIES

Kim Harris
FORTHRIGHT ENTERPRISES
P.O. Box 50911
Palo Alto, CA 94303

In true ideal FORTH programming style the definitions contained within the screens clearly designates their use.

```
81 LIST 82 LIST
SCR # 81
 0 ( Software Development tools  fig-FORTH 1.x )
 1 ( for LOADing )
 2 : LOAD   ( scr# — )   DUP .   LOAD   ;
 3 : THRU   ( 1stScreen# lastScreen# — )
    ( LOADs range of screens )
 4    1+ SWAP DO    I LOAD   LOOP   ;
 5
 6 ( non-destructive stack print )
 7 : DEPTH   ( — #StackCellsUsed )   SP@
    S0 @  SWAP - 2/ ;
```

The following utility indexes 10 screens at a time and is an excellent aid in searching.

```
HEX : +INDEX  113  0  DO
DUP 10+ SWAP OVER INDEX
KEY ?ESC IF LEAVE THEN LOOP;
```

The following utility was contributed by Sam Bassett and is an excellent program development aid that shows you what the current base is

```
: BASE?
   BASE @
   DUP           -
   DECIMAL
   .
   BASE !
;
```

Here is an adaptation of George Shaw's VIEW to use the word WHERE, which on my system invokes a full screeen editor that highlights the word pointed to by a block number and displacement. It certainly helps pick out a word in dense code.

SCR # 66

```
 0 ( 'VIEW' USING 'WHERE'    4/15/81      R.E.E. )
 1 ( ADAPTED FROM FORTH DIMENSIONS VII,
    NBR 6, P 162  )
 2 FORTH DEFINITIONS
 3 : >DOC< BLK @ , IN @  , ;  ( SAVE BLK AND
    DISPLACEMENT     )
 4 : CONSTANT  >DOC< [COMPILE] CONSTANT  ;
    ( REBUILD THE WORDS   )
 5 : VARIABLE  >DOC< [COMPILE] VARIABLE  ;
    ( THAT BUILD WORDS.  )
 6 : :         >DOC< [COMPILE] :         ;
 7 : ;CODE     >DOC< [COMPILE] ;CODE     ;
 8 : USER      >DOC< [COMPILE] USER      ;
 9 : CREATE    >DOC< [COMPILE] CREATE    ;
10 : VIEW  [COMPILE]  ' NFA  ( GET HEAD OF
    THE HEADER )
11         DUP              ( COPY THE ADDRESS )
12         2 - @       ( GET THE DISPLACEMENT )
13         SWAP 4 - @    ( GET THE BLOCK NUMBER )
14         WHERE          ( GO SHOW & HIGHLIGHT ) ;
15    ;S
```

HELP WANTED

Senior Level FORTH Programmers

Friends-Amis
505 Beach Street
San Francisco, CA 94133
Call: Tom Buckholtz
(415) 928-2800

Intermediate & Senior Level FORTH Programmers for Data Entry Applications

MSI Data
340 Fischer Avenue
Costa Mesa, CA 92627
Call: Joan Ramstedt
(714) 549-6125

PRODUCT REVIEW

by C.H. Ting, Feb. 26, 1981

Timin-FORTH, from Mitchel E. Timin Engineering Co., 9575 Genesee Ave., Suite E2, San Diego, CA 92121, (714) 455-9008. 8" single density diskette, $95.00

I was invited by Dr. Timin to compare his CP/M FORTH (FD II/3, p. 56) with the Z-80 FORTH by Ray Duncan, Laboratory Microsystems (FD II/3, p. 54; FD II/5, p. 145) I ran the two FORTH systems on his home made Z-80 computer (S-100 bus, 6 MHz) The results of a few bench marks were:

Program	Timin	Z-80
: LOOPTEST 7FFF 0 DO LOOP ;	2.3 sec	2.9 sec
: -TEST 7FFF 0 DO I DUP - DROP LOOP;	5.9	7.4
: *TEST 7FFF 0 DO I DUP * DROP LOOP ;	44.0	54.9
: /TEST 7FFF 0 DO 7FFF I / DROP LOOP ;	74.3	88.6
: WIPE 120 61 DO I CLEAR LOOP ;	34.3	81.8
97 LOAD (four hundred eighty 9's)	17.9	18.6

I was surprised that Timin-FORTH which is 8080 fig-FORTH ran faster than Z-80 FORTH which uses the extra Z-80 registers for IP and W. Dr. Timin's opinion was that the Z-80 instructions using these extra registers are slower then the simpler 8080 instructions. The word WIPE tests disc access time. Timin-FORTH accesses the disc by 1024 byte blocks, and it is twice as fast as Z-80 FORTH, which reads/writes by 128 byte sectors, as in the fig-FORTH model.

The dictionary in Timin-FORTH is about 11 Kbytes, including an editor and an assembler. The editor is the same as that of the fig-FORTH model. The assembler has all the Z-80 instructions. An interesting word SAVE allows the whole system including application words to be preserved

as a CP/M file which can be loaded back for execution. It maintains eight 1 Kbyte disc buffers.

The documentation supplied with the system is a 68 page booklet 'USER'S MANUAL & TUTORIAL'. It is a very well done manual introducing users to the systems and to the FORTH language. However, source listings are not provided.

My overall impresssion was that this is a well rounded FORTH system suitable for engineering and professional applications.

Editors Comment -- FORTH Dimensions refrains from publishing timing benchmarks as this reflects processor speed more than effectiveness of problem solving. However, the above review points out that the allegedly superior Z-80 runs these tests slower than the 8080. Our point is that the user should evaluate all aspects of problem solving: hardware characteristics, language implementation and application technique. The Timin manual is sold separatly for $20.00. This price is not justified by the copy received for our evaluation.

HELP WANTED

FORTH PROGRAMMER

PDP-11 RSX Op Sys On Site Contractor

Micro/Temps
790 Lucerne Dr.
Sunnyvale, CA 94086
(408) 738-4100

FORTH TELE-CONFERENCE IS NOW OPERATIONAL

FORTH now has a dynamic, public access data base. By dialing into the FIG CommuniTree (tm, the CommuniTree Group) you may access our tele-conferencing system. It was created by Figger John James to allow group interaction to build upon our collective knowledge.

The number is 415-538-3580. The system runs 24 hours a day. Use a 300 baud modem and start with two "returns", the system is self-instructing. This conference holds information on employment, vendors, applications, announcement calendar, inquiries, books, etc. Information of the conference is organized in a tree structure, hence the name " Conference Tree".

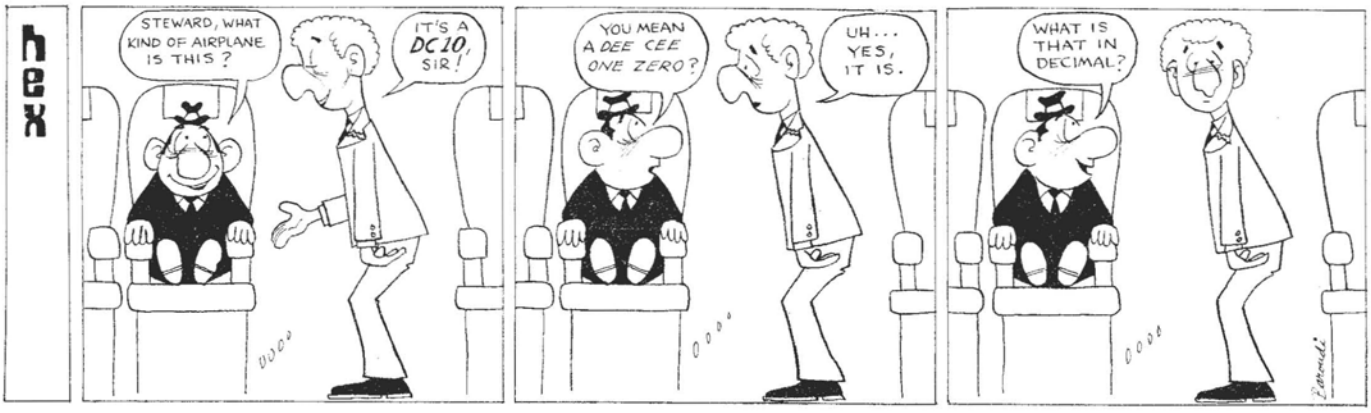
Our hope is that half of the callers will review the available material and then ask questions. The other half should add answers to these questions. You simply find a topic or message and attach your query/response. Users naturally organize their material in a form that facilitates retrieval.

This system was written in Cap'n Software Version 1.7. Versions for other than Apple II are being developed.

For availability contact:

The CommuniTree Group
Box 14431
San Francisco, CA 94119

or call the original Tree: (415) 526-7733.



FORTH, INC. NEWS

MAJOR EXPANSION PLANS

FORTH, Inc. is now entering a major expansion phase, according to President Elizabeth Rather. Appearing on a panel on "Programming Languages for Small Systems" at the recent NCC in Chicago, Rather observed, "The level of excitement and enthusiasm about FORTH in the industry is tremendouus. We are increasing our number of OEM'S and we have been approached by several major silicon manufacturers desiring to obtain marketing rights for special versions of polyFORTH. Arrangements are also being made to produce the FORTH processor, and we expect this project to start very soon."

LIFEBOAT REPRESENTATIVE VISITS

Masa Tasaki, Managing Director of Lifeboat, Inc., FORTH, Inc.'s distributor in Japan, spent two days at FORTH, Inc. recently to discuss mutual marketing plans. Lifeboat, Inc. is one of the few software distributors in Japan, and polyFORTH is the top of their product line. Tasaki has installed over 40 polyFORTH systems in Japan in the past year, and plans to sell an additional 50 polyFORTH systems by the end of 1981.

STARTING FORTH BOOK PREPRINTS AVAILABLE

STARTING FORTH, a 380-page book introducing the FORTH language and operating system will be published by Prentice-Hall this September in both hard and soft-bound editions. FORTH, Inc. is offering limited preprints to customers until then. The preprint, numbered and signed by both author Leo Brodie and Charles H. Moore sells for $50.00 (plus 6% sales tax for residents of California). You may reserve a copy of STARTING FORTH by calling Winnie Shows at (213)372-8493. All orders must be pre-paid.

RECENT FORTH COMMERCIAL APPLICATIONS

Work has just been completed for Raytheon Corporation on a terminal cluster (up to 32 terminals with a single concentrator). Each component of the system is controlled by an 8085 processor, and all are programmed independently, using polyFORTH. This is a capability they've never had before -- to do custom programming and provide extensibility. Terminals up to two miles away can be polled at a rate 30 times faster than the previous protocol, which was written in assembler. Dean Sanderson was the principal programmer on the project.

The famous 200" Hale Telescope at Mt. Palomar Observatory (near San Diego) has recently installed a polyFORTH system for data acquisition and analysis using a PDP11/44 and a Grinnell display processor. The Observatory has been using FORTH since the early 1970's, including a miniFORTH system installed in 1975 and an early polyFORTH installed in the late 70's. Barbara Zimmerman, a programmer at Cal Tech (which operates the observatory) said, "I am extremely impressed by the level of polish and sophistication in polyFORTH, and the performance of this system is outstanding." The type of work done involves reading data from an 800 x 800 array of CCD sensors, integrating and recording the data, and displaying it in the Grinnell. Charles Moore installed the system, which features a comprehensive math package for analysis as well as basic image-processing functions.

A by-product of this installation is the availability of polyFORTH in RK05 disk cartridges. These are available with on-site installation.

SCHEDULE OF UPCOMING FORTH, INC. SEMINARS AND WORKSHOPS:

Location	**Seminar**	**Workshop**
Palo Alto	June 4	June 5
Houston	July 7	July 8
Tampa	July 9	July 8
Irvine	July 23	July 24

PARAMETER PASSING TO DOES>

David McKibbin
Sygnetron
2103 Greenspring Drive
Timonium, MD 21093

Often in programming one runs into the case where several different processes share similar structures. Not wanting to waste time or space for redundant code, the programmer usually creates a subroutine or procedure to execute the basic structure. Then the individual processes merely pass arguments to the prodecure to accomplish their task. Several schemes can be used to pass these parameters. In simple cases, the stack can be used directly. This is the typical act of programming in FORTH.

```
: DELAY  0 DO LOOP ;        ( SPIN FOR A WHILE )

100 DELAY                   ( COUNT PASSED ON THE STACK )
```

However, as the procedures get more complex it gets more and more difficult to keep track of the passed parameters especially when the procedure itself is using the stack heavily. Also many times it is necessary to pass not only numbers but operators or words as parameters. One means of accomplishing this is via <BUILDS DOES>. Parameters will be stored in the parameter field of the newly defined word and accessed from DOES> via a new word { $ } . 1 $ will push the first parameter on the stack, 2 $ will push the second, etc. All parameters are 16 bits. Variable R# is used to store the parameter base address.

```
: $  1 - DUP + R# @ + @ ;     ( PUSH THE N'TH
     PARAMETER )

: EXAMPLE <BUILDS DOES> R# ! 1 $ 2 $ EXECUTE ;

EXAMPLE ZZZ 90 , ' EMIT CFA ,  ( TYPE A "Z" )

EXAMPLE SPC 10 , ' SPACES CFA , ( TYPE 10 SPACES )
```

Now that the mechanics are explained the following example will more fully demonstrate its usage. Both DUMP (16 bit dump) and CDUMP (8 bit dump) share a common structure with only a few inner words differing. DUMPS is a new defining word used as a procedure for both DUMP and CDUMP.

```
: U.R  0 SWAP D.R ;

: DUMPS <BUILDS DOES> R# ! ( STORE PARAMETER BASE
ADDRESS )
    BASE @ R> HEX          ( SAVE BASE AND SET HEX )
    OVER + SWAP            ( CONVERT TO BEGINNING
  AND END ADDRESS )
    BEGIN
      CR  DUP 4 U.R  2 SPACES  ( TYPE ADDRESS )
      1 $ 0 DO
        DUP 2 $ EXECUTE  3 $ U.R  4 $ +
        2DUP =  OVER 16 MOD 0=  OR IF
          LEAVE THEN
      LOOP
      2DUP =  ?TERMINAL OR
    UNTIL
    DROP DROP CR
    R> BASE ! ;            ( RESTORE BASE )

DUMPS CDUMP  16 , ' C@ CFA , 4 , 1 ,
  ( 2-ADDRESS, 1-COUNT )
DUMPS DUMP  8 , ' @ CFA , 6 , 2 ,
  ( 2-ADDRESS, 1-COUNT )
```

What has been accomplished is akin to passing procedures/functions as parameters in Pascal. I expect that there are other ways to do this FORTH beyond what has been proposed.

FIG-FORTH UNDER OS-65U

Software Consultants has announced the availability of Fig-FORTH under OS-65U for the Ohio Scientific Line. The package includes assembler and a terminal oriented editor and is available now for $79.95.

This version is said to support hard-disk, multi-user systems and may even be run in one partition and BASIC in another.

For more information contact:

Software Consultants
7053 Rose Trail
Memphis, TN 38134
(901) 377-3503

COMPILER SECURITY

George W. Shaw III
SHAW LABS, LTD.
17453 Via Valencia
San Lorenzo, CA 94580

How it Works and How it Doesn't (Adapted from a section of the Acropolis A-FORTH manual)

There is much argument about parameter validation and error detection in FORTH. Many problems exist with many good solutions. Fig-FORTH and its derivitives have taken one route of extensive protection in compiler directives and their associated words. This is not an only solution in this area. Its extensiveness may not be necessary. There may be better alternatives. Read on, learn how fig-FORTH works, consider the options and then decide. Your opinion and ideas are needed.

Fig-FORTH and its derivitives provide a type of compiler error detection referred to as "compiler security". Compiler security provides protection against structural programming errors made by the programmer as well as insuring the proper machine state and, in a very few instances, the validity of parameters. Though it depends on the type of programming, the most common errors are structural errors* , machine state errors, and then parameter errors, respectively.

(* structural errors may be caught internally by detecting parameter errors. See text.)

STRUCTURAL ERRORS

The compiler security system uses two methods to trap structural programming errors inside of colon-definitions. Structural errors are those caused by incorrect program structure; either improper nesting of structures or not completing a structure inside of a definition. Either of these conditions would cause the program to compile incorrectly and could cause disastrous effects (i.e. a system crash) at run-time. The methods used by the compiler security system entail either checking a value on the top of the stack (to verify the proper nesting of structures) or checking that the stack position is the same at the end of a definition as it was at the beginning of the definition (to ensure program structure completion). These two methods probably trap about ninety percent (90%) of the structural programming errors that a programmer might make.

The first in each of the paired structural compiler directives (i.e. pairs such as IF THEN , DO LOOP , etc.) leave on the stack at compile time a value which is checked by the ending structure to ensure the proper nesting of structures. For example the word IF leaves, in addition to the other data necessary to compile an IF , the value of two (2) on the top of the stack. The words ELSE and THEN remove a value from the top of the stack and check to see if it is a two (2). If the value on the stack was not a two (2), a Conditionals Not Paired error (#19) results, and compilation is terminated (control returns to the keyboard). If the value is a two (2) the remainder of ELSE or THEN executes, removing the necessary data from the stack to finish the structure, and compilation continues on to the next word.

Below is a table of the conditional pairs for the current structural compiler directives, with the values placed on the stack open and the values removed from the stack in parenthesis. Note that UNTIL and END as well as THEN and ENDIF have the same effect. Only the former of each pair are presented here for clarity.

```
BEGIN  1     UNTIL   (1)
BEGIN  1     WHILE    4      REPEAT  (1) (4)
BEGIN  1     AGAIN   (1)
IF     2     THEN    (2)
IF     2     ELSE    (2) 2  THEN    (2)
DO     3     LOOP    (3)
DO     3     +LOOP   (3)
DO     3     /LOOP   (3)
DO     3     +/LOOP  (3)
```

Note that ELSE tests and replaces the same value on the stack. Because of this the current compiler security system cannot detect the presence of multiple ELSEs in a definition. For example, in the definition:

```
: ELSE-EXAMPLE    flag --- true or false message )
  IF  ." True part 1 "  ELSE  ." False part 1 "
    ELSE  ." False part 2 "  ELSE  ." False part 3 "
  THEN  ;
```

if compiled , (and it will compile,) and then executed with a boolean value (zero or non-zero) on the stack, will execute without crashing the system. But the execution may not be what you expected. If entered with a true flag (non-zero) the "True Part 1" and the "False Part 2" will print, while if entered with a false flag (zero) the "False Part 1" and "False part 3" messages will print. To borrow a phrase from Kim Harris, probably "Not what you had in mind!".

This is the only case I know of where the compiler security system plainly does not work, but there are probably more.

How is this, apparently incomplete, structure checking performed? Read on.

The values on the stack is verified by ?PAIRS . For example the words ?PAIRS , BEGIN and AGAIN are defined as follows:

```
: ?PAIRS  -  19  ?ERROR ;
: BEGIN  ?COMP  HERE  1 ;  IMMEDIATE
: AGAIN  1  ?PAIRS  COMPILE  BRANCH  BACK
;  IMMEDIATE
```

BEGIN first checks to make sure that it is being executed in compile mode (inside a definition) with ?COMP which issues an error if it is not. It leaves the current dictionary address on the stack (HERE) as a branching reference for AGAIN , and then the 1 as the first of a conditional pair. When AGAIN later executes during the compilation of the definition it first checks the stack to see that a BEGIN preceded it at the same level of nesting by executing ?PAIRS . ?PAIRS expects to find a matched pair of values, in this case ones (1), as a matched set of conditional pairs. If ?PAIRS does not find a matched set, it aborts with a Conditionals Not Paired error (#19). If the values on the stack are paired, it removes them and returns.

The above simple form of error checking is very effective, but as structures become more complex, manipulating and maintaining the stack values can become cumbersome and unwieldy. The above is also not yet complete. One more check must be executed to ensure that the structures in the definition have been completed. Since the above error checking leaves data on the stack if a structure has not been completed, the simplest check is that of the stack position. When a definition is entered : (colon) stores the Current Stack Position in the user variable CSP . At the end of a definition, ; (semi-colon) executes ?CSP to compare the current stack position to the value stored in CSP . If the values differ a Definition Not Finished error (#20) occurs indicating that either data was left on the stack or that too much data was removed from the stack, i.e. that a programming structure was probably not completed. The word "probably" is used here because other conditions, such as the improper or sometimes various proper uses of the word LITERAL , will cause the same error condition to occur.

MACHINE STATE ERRORS

The loading and execution of a FORTH program causes the system to enter several different machine states. Three of these are loading, compiling, and executing. Each of these states is defined by its own set of parameters and some states may even overlap. For example, while loading a screen off the disk the machine will be either executing or compiling. Here the loading state has overlapped with either the execution or compilation state. The machine cannot be in the execution state and the compilation state at the same time, though the states may be interleaved. An example of interleaved states is the use inside a definition of a program segment similar to this:

[SCREEN 3C0]+ LITERAL

which temporarily suspends compilation to calculate the value within the brackets and then compiles it as a sixteen (16) bit literal. Remember though, that to compile, the machine is executing a program, and that compiler directives (such as LITERAL above) execute during compilation to perform their task, but the machine state remains that of compilation.

Certain words require that the machine be in a specific state to execute properly. These words are programmed to contain one of the following words:

?COMP ?EXEC ?LOADING

which check for their corresponding state and issue an error message if the machine is not in that state. Below is a description of each of the above words and the parameters which determine the current machine state.

?EXEC or ?COMP

The execution state or compilation state is determined by the value of the user variable STATE which has a zero (0) value if the machine is in the execution state and a non-zero value the machine is in the compilation state.

?LOADING

Loading is determined when the value of the user variable BLK has a non-zero value. A value of zero for BLK indicates that input is coming from the user's terminal and that the machine is therefore not loading.

The above words are defined as follows:

```
: ?EXEC   STATE  @  18  ?ERROR ;
: ?COMP   STATE  @  0=  17  ?ERROR ;
: ?LOADING BLK  @  0=  22  ?ERROR ;
```

If the machine is not in the execution state when ?EXEC executes an Execution Only error (#18) occurs.

If the machine is not in the compilation state when ?COMP executes a Compilation Only, Use in Definition error (#17) occurs.

If the machine is not in a state of loading when ?LOADING is executed a Use Only When Loading error (#22) occurs.

The testing of machine states as above is necessary when words such as BEGIN and AGAIN (see example in STRUCTURE ERRORS above) are used. These words may only be compiled because they must compile something other than themselves which is not known at the time they are executed.

PARAMETER ERRORS

During compiling and similar operations there are only a few parameters which are actually checked. In most cases, the parameters checked are those involved in the other areas of compiler security or those which deal with the size or validity of the dictionary and stack.

The words involved in other compiler security areas are !CSP , ?CSP , ?PAIRS . These words are used to protect against structural programming errors as described above in STRUCTURAL ERRORS. An explanation of each of the uses of these words is as follows:

!CSP ?CSP

These words are used together to check for changes in the stack position. !CSP stores the current stack position in the user variable CSP . ?CSP compares the value in CSP to the current stack position and, if they are not the same, issues a Definition Not Finished error (#20). !CSP and ?CSP are currently used in : and ; respectively to ensure that all structures in the definition have been completed before the semi-colon. Any structures uncompleted will leave data on the stack and thus allow ?CSP to flag the error. These words can also be used to check the stack effect of user definitions. For example, if a definition should have no stack effect (leaves the same number of items on the stack as it removes) the following would test this:

```
!CSP    cccc    ?CSP
```

which would execute a definition named cccc and issue a Definition Not Finished error (#20) if the number of items on the stack at the beginning and end of the definition were different.

?PAIRS

This word is used when testing for correct structure in compiler directives (see STRUCTURE ERRORS) to check that the value of the two numbers on the stack is the same. If the value of the two compilation conditionals on the stack is not the same, a Conditionals Not Paired error (#19) occurs. ?PAIRS can be used to test similar situations in user programs, but the error message given will be the same (error #19).

The checks on the dictionary and stack consist of testing the stack for underflow, the dictionary and stack for overflow, and the name of the dictionary entry to be created for uniqueness (in A-FORTH this test is optional and there is a test to ensure that a definition name is not null). Some of the tests are performed during the execution of other functions by the testing word (such as the tests performed by WORD and by CREATE). Only the testing performed by these words will be described here.

CREATE

This word creates a dictionary header for a new word. In the process of creating this header a dictionary search is performed to check that the header is unique. The message given if a duplicate is found is Isn't Unique (#4). This is not a fatal error but just a warning. A-FORTH allows the disabling of this test (and the associated message) and performs another test for a dictionary entry whose name is a null. The creation of a dictionary entry with a null name is not allowed because the null is the name of the entry interpreted at the end of the disk or terminal buffers. If an attempt to create a null entry is detected a Null Definition Name! error (#9) is given. If a dictionary entry with a null name were created, the system would attempt to interpret this as the end of the current buffer with unpredictable results.

?STACK

This word checks that the parameter stack is within bounds. It compares the current stack position (by executing SP@) against the base stack position in user variable S0 to check for a stack underflow. It also checks that there are at least 128 bytes of dictionary space left (to leave room for PAD and stack work). If the stack underflows an Empty Stack error (#1) is given. If the stack comes within the 128 bytes of the dictionary a Full Stack error (#7) is given. ?STACK is not executed at runtime unless compiled by the programmer, though it is executed frequently during compiling and text interpretation.

WORD (A-FORTH only)

This definition moves text from the current input buffer to the head of the dictionary. The error test performed checks that there is enough space between the head of the dictionary and the top of the stack for the text about to be moved. If there is not enough space a Dictionary Full error (#2) is given. This prevents the system from crashing by writing over its own stacks.

DO WE NEED IT?

Should we have all this security all the time? Or just when we think we need it? Fig-FORTH currently does not give us a choice on the matter. Sure, we can compile on top a new set of compiler directives which don't have the tests, but we have then already wasted all the memory for the secure directives, the ?XXX words, and the lot. The reverse course I consider more appropriate. The kernel system should have as little protection as possible. The system should not suffer the overhead for those who do not desire it. If security is desired, a "Novice Programmer Protection" package could be

compiled into a user's area which would include all the words necessary to protect him or her (and the other users) from him or herself. This would allow protection even for the words such as ! (store), FILL and CMOVE when desired.

Something as simple and extremely effective as the !CSP and ?CSP in : and ; respectively may be left in the kernel system to give warning to even the best of us when necessary. Definitely, also the stack checks at compile time and possibly the uniqueness (though it should be optional) and null definition (currently A-FORTH only) checks should be left in, but the structure and state testing is often incomplete and annoying. Anyone who has tried to write and secure a good general CASE structure, or a BEGIN WHILE REPEAT loop which allows multiple WHILEs will know what a pain it is to try to secure them in a reasonably complete fashion. For these people compiler security dosn't work. Additionally, new structures transported from my system to another may not remain secure because the same conditional pair numbers used in my structure on my system may have been used in a different structure on the other system. Again, the compiler security dosn't work.

The same method used in high level structure testing is also used in one known assembler, which the author considers totally inappropriate. If one is programming in FORTH assembler one is doing so for speed, which may require not being structured at all.

Currently, the matter of compiler security is being studied by the group writing the next 8080 fig-FORTH version (which could possibly outline a new model). Should we have all the protection all the time, or just some of it and a programmer protection package? Or maybe there is a better alternative. Your input is wanted and needed. Write to the 8080 group at FIG, PO Box 1105, San Carlos CA 94070 and tell us what you think.

NEW PRODUCTS

POLYMORPHIC FORTH NOW AVAILABLE

FORTH is now available for the PolyMorphic Systems SSSD 5" systems (8813 & 8810). The PolyMorphic disk operating system has been patched in and the system is interfaced to the PolyMorphic operating system. PolyMorphic FORTH includes a modified systems disk, and brief documentation on changes to interface to the PolyMorphic SSSD 5" disk operating system -- based on 8080 Fig-FORTH. Price is $50.00. For more information contact:

Ralph E. Kenyon, Jr.
ABSTRACT SYSTEMS, ETC.
145-103 S. Budding Avenue
Virginia Beach, VA 23452

FORTH FOR HP83/HP85

A disk based FORTH is now available for the HP85/HP83 personal computers. The implementation is the FIG FORTH 1978 standard with some machine dependent utilities. User receives both 16k and 32k versions with user space being 2k and 18k respectively. Both versions require a disk. Included is an assembler, a FORTH decompiler and editor. This is not an HP supported product but available through the user's library. FORTH, in object form (no source), an assembler, decompiler and editor, in source, are sent on a disk. This product recommended for experienced users only! Those familiar with FORTH should have no trouble using this system (i.e. there is no manual inluded). However, sufficient references are given. Current cost is $50.00. For more information contact Nany Reddington at (503) 757-3003.

FORTH PROGRAMMER AVAILABLE

3 mos. experience with FORTH (also know BASIC & COBOL) Active member of F.I.G. Contact: Martin Schaaf, PO Box 1001, Daly City, CA 94017 (415)992-4784

USERSTACK

Peter H. Helmers
University of Rochester
Department of Radiology
Medical Center, Box 648
Rochester, NY 14642

INTRODUCTION

One of the advantages of FORTH is its use of a stack oriented architecture. In conventional FORTH implementations, one has available two kinds of stacks: the return stack and the parameter stack. In general, the return stack is used to keep track, at execution time, of the path of invocation of nested FORTH words while the parameter stack is used to manipulate data used within and/or passed between FORTH words.

Unfortunately, in the real world, such a clean segmentation between parameter data and execution nesting data tends to break down. For example, DO...LOOPs are implemented by using the return stack to keep track of the loop count and associated data. The motivation for this violation of the sanctity of the return stack with DO...LOOP parameters is the desire to separate the DO...LOOP data from any parameters being used by the programmer within the loop. Failure to do so would allow confusion of loop parameters with actual user data -- causing a consequent abnormal execution of the DO...LOOP arising from an unwarranted modification of loop parameters.

In addition to the above saving of DO...LOOP parameters on the return stack, it is not uncommon practice for a programmer to want to save some parameter stack data in order to be able to first calculate using data beneath it. One previously employed method to do this was to temporarily push the parameter stack data onto the return stack, and then later retrieve it when subsequently needed. Admittedly, this is an easy -- lazy! -- way to achieve tranisent data storage. But woe unto those who forgot to pop the return stack of this temporary data...!

Editor's Note: Mr. Helmers uses URTH, a dialect of FORTH.

USER STACKS

The "user" stack concept allows a FORTH program to retain the convenience of an auxiliary stack, but in such a way as to avoid mixing temporary data with execution time return information. As an added convenience, this concept allows creation of multiple, named, stacks which can be typed according to the number of (two byte) words per stack element.

A user stack can be thought of as an array (integer, double precision, or real) of data which has implicit addressing. Consider, by way of analogy, a conventional array such as:

```
100 ()DIM MY-ARRAY
```

One would store the 53rd integer element by explicitly stating the index:

```
52 MY-ARRAY !   ( ZERO ORIGIN...)
```

This would take data from the top of the parameter stack and store it in MY-ARRAY. Alternatively, one would access an integer from this array by:

```
27 MY-ARRAY @
```

The disadvantage of arrays is that they require both an explicit index, and an explicit load (@) or store (!) operator. While an array could be used for temporary storage of parameter stack data, such programming practice is not neccessarily clear or efficient.

So how does a user stack help us? Consider the integer user stack defined:

```
100 STACK MY-STACK
```

MY-STACK would, in this case, have a size of 100 integer elements. Data can be put into this user stack from the top of the parameter stack by:

PUSH MY-STACK

while it can be retrieved back to the parameter stack by:

POP MY-STACK

Note that addressing is implicit-- there are no indices -- and that the direction of data transfer is set by the PUSH and POP words.

USER STACK WORDS:

In practice, three types of user stacks have proved useful; STACK, DSTACK, and FSTACK. While stack variables created by these three defining words all use the PUSH and POP words to save and retrieve data, the amount -- or type -- of data pushed or popped differs. As discussed earlier, STACK deals with integer (two byte) words. DSTACK consists of elements of double precision integer words (four bytes) while FSTACK elements are floating point numbers (six bytes). All three of these words are defined in terms of an arbitrary n-precision NSTACK word which allows specification of any number of two byte words per stack element.

Two other words are also useful with user stacks. There are EMPTY-STACK and ?STACK. Note that both of these cannot (presently) be used within colon definitions. The line:

EMPTY-STACK MY-STACK

will, for example, reset the stack pointer for the user stack: MY-STACK so that it will be empty. Again using the MY-STACK example,

?STACK MY-STACK

will dump out the contents of the stack from the top of the stack through the bottom of the stack. ?STACK is intended purely as an aid in debugging.

IMPLEMENTATION:

As was previously mentioned, STACK, DSTACK, and FSTACK are all defined in terms of a more general NSTACK defining word. A line such as:

22 4 NSTACK WIDE-STACK

will define a 22 element stack with eight bytes (four words) per element. NSTACK has two primary parts. The first part, executed when a new stack is defined, builds a FORTH word header, stores some stack definition parameters into the dictionary, and finally allocates the actual dictionary space for the stack, The second part, written in 8080 assembly language for speed, defines the execution time actions taken by the stack variable. Both of these defining parts will be explored in greater detail below.

The format of the user stack in the dictionary is shown in Fig. 1. It consists of a normal FORTH header, followed by the following four stack definition parameters:

a) current stack pointer (two bytes)
b) #words per stack element (one byte)
c) maximum stack pointer address (two bytes)
d) minimum stack pointer address (two bytes)

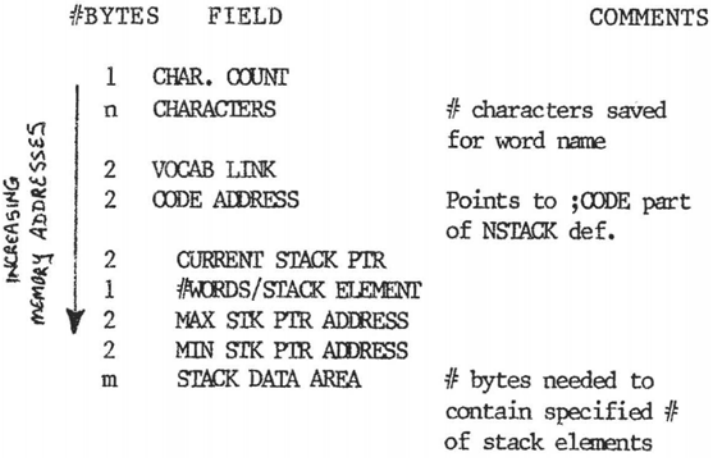


Figure 1 -- Dictionary Layout for a Stack Type Variable

```
*** BLOCK #  150

   ( STACK DATA TYPES -- PHH 23 OCT 80 )
   ( [#ELEMENTS][#WORDS/ELEMENT] NSTACK NNNN CREATES THE STACK)
   (    CALLED NNNN WITH THE GIVEN NUMBER OF ELEMENTS. STACK DATA)
   (    TYPES MAY BE PUSHED OR POPPED TO OR FROM THE PARAMETER )
   (    STACK )
   : NSTACK <BUILDS SWAP OVER DUP + * DUP ( #BYTES IN STACK )
      HERE + DUP , ( SET UP CURRENT STK PTR )
      ROT C, ( #WORDS/STK ELEM ) , ( MAX SP ADDR )
      HERE , ( MIN SP ADDR ) 10 + ALLOT ( SPACE FOR STACK )
   ;CODE  WPARAM 2 + B LXI, B DAD, H PUSH, ( HL PTS TO #WRDS/ELE )
      RP LHLD, H DCX, D M MOV, H DCX, E M MOV, RP SHLD, ( SAV IP )
      H POP, B POP, ( PUSH/POP FLAG ) C A MOV, B ORA, 0=,
      IF, ( POP DATA FROM NSTACK TO PARAMETER STACK )
         M A MOV, ( #WRDS/ELE ) H DCX, M B MOV, H DCX, M C MOV,
         ( CSP IN BC )
   -->


*** BLOCK #  151

   ( NSTACK DEFINITION CON'T )
         BEGIN, ( MOVING DATA, WORDS AT A TIME ) PSW PUSH,
            B LDAX, B INX, A E MOV, B LDAX, B INX, A D MOV,
            PSW POP, ( COUNT ) D PUSH, ( DATA FROM NSTACK )
            A DCR, 0=,
         END,
      ELSE, ( PUSH PARAMETER STACK DATA TO NSTACK )
         M A MOV, H DCX, M B MOV, H DCX, M C MOV, ( CSP IN BC )
         BEGIN, ( TO PUSH DATA ) D POP, PSW PUSH, ( COUNT )
            B DCX, D A MOV, B STAX, B DCX, E A MOV, B STAX,
            PSW POP, A DCR, 0=,
         END,
      THEN,
      C M MOV, H INX, B M MOV, ( SAVE NEW CSP )
      RP LHLD, M E MOV, H INX, M D MOV, H INX, RP SHLD, NEXT JMP,
   -->


*** BLOCK #  152

   ( STACK TYPE VARIABLES CON'T )
   CODE POP   0 H LXI, PUSH JMP,
   CODE PUSH -1 H LXI, PUSH JMP,
   : STACK  1 NSTACK ;   ( INTEGER ELEMENTS )
   : DSTACK 2 NSTACK ;   ( DOUBLE PREC INTEGER ELEMENTS )
   : FSTACK 3 NSTACK ;   ( FLOATING POINT ELEMENTS )

   ( USER AIDS WHICH CANNOT BE COMPILED ... )
   ( EMPTY-STACK NNNN -- EMPTIES THE USER STACK "NNNN" BY
   (    RESETTING ITS STACK POINTER )
   : EMPTY-STACK ]'[ DUP 3 + @ SWAP ! ;
   ( ?STACK NNNN -- PRINTS OUT THE CONTENTS OF THE USER STACK )
   : ?STACK ]'[ DUP 3 + @ OVER @ - DUP 0> IF 1 ->L SWAP @ DUMP
      ELSE 2DROP T" USER STACK EMPTY " THEN ;
   ;S
```

```
OK
OK
OK
OK 100 STACK MY-STACK
OK 35 STACK YOUR-STACK
OK
OK 11 22 33 44 55 66 77 88 99
OK PUSH MY-STACK
OK PUSH MY-STACK
OK PUSH YOUR-STACK
OK PUSH MY-STACK
OK PUSH MY-STACK
OK PUSH MY-STACK
OK PUSH YOUR-STACK
OK + POP MY-STACK - .
-11  OK
OK POP YOUR-STACK POP YOUR-STACK 2DUP . . + .
 77   33   110  OK
OK POP MY-STACK .
 55  OK
OK POP MY-STACK .
 66  OK
OK POP MY-STACK .
 88  OK
OK POP MY-STACK .
 99  OK
OK
OK
OK
OK
OK
OK
OK ?STACK MY-STACK
USER STACK EMPTY OK
OK 11 PUSH MY-STACK 22 PUSH MY-STACK ?STACK MY-STACK
3A7A   0016 000B
OK
OK
OK EMPTY-STACK MY-STACK ?STACK MY-STACK
USER STACK EMPTY OK
OK
OK
OK
OK
```

Note that the stack, consistant with the 8080 architecture, grows down in memory. Following these stack parameters is the actual stack area which is allocated in the dictionary.

The PUSH and POP words are code definitions (for speed) which push a 0 or -1 flag value to the top of the parameter stack. Thus, when the stack variable is subsequently executed, this flag is used to differentiate between popping from the user stack (flag=0) and pushing to the user stack (flag=1). The assembly code is thus separated into two very similar execution loops which move stack data one word at a time until the proper number of words for the stack element have been moved; these two loops differ only in the direction of the data transfer. In both loops, the A register contains the current word count which is intially set to the number of words per stack element and decremented each time through the loop. The BC register pair contains the current user stack pointer while the HL register pair contains the address of the stacks parameter field so that the new user stack pointer value may be saved after all words within the stack element have been transferred.

CONCLUDING REMARKS

These user stacks have been optimized to provide rapid execution speed at the expense of high level transportability and error checking for a stack pointer out of bounds. It is felt that the concept, in whatever realization, is important since it provides a very readable and structured method to temporarily store and sort data without having to resort to such "unclean" practices as using either explicitly addressed arrays or the return stack. It's the type of FORTH word that, once you have it, prompts the question: "it's so obvious, why didn't someone think of it before?"

NEW PRODUCT

STAND-ALONE FIG-FORTH FOR OSI

FORTH Tools has announced stand-alone Fig-FORTH for all OSI mini-floppy computers that combines Fig-FORTH with stand-alone machine drivers by FORTH Tools. With this system OSI-65D is superfluous--with FORTH booting up directly, yet the disk is OS-65D compatible.

Since FORTH Tools FORTH dispenses with the OSI operating system, FORTH Tools has developed disk, display and keyboard drivers for the OSI hardware.

FORTH Tools FORTH for OSI is strictly compatible with Fig-FORTH. All words in the Fig model, including disk support, work correctly. Portability to other machines is also claimed.

Stand-alone Fig-FORTH for OSI is available on one 5-1/4" disk for C1 (Superboard), C2 and C4 machines with 24K. Product includes a structured 6502 macro-assembler and disk utilities designed by FORTH Tools and the FIG portable line editor. Complete technical documentation and the fig-FORTH glossary are also included. The complete price is $49.95. For more information contact:

FORTH Tools
Box 12054
Seattle, WA 98102

DEA- EDI---

I AM AFR--- THA- THE LET--- IN THE LAS- ISS-- ABO--
FOR-- INC- USI-- ONL- THR-- LET--- NAM- FIE--- HAS
HAD THE OPP----- EFF--- FRO- WHA- THE WRI--- WAN---

HIS LET---(LIK- THI- ONE) SHO-- THA- SAV--- ONL-
THR-- LET---- AND COU-- IS JUS- ABO-- OPI---- IN
TER-- OF A TRA-- OFF BET---- SAV--- MEM--- AND
KEE---- LEG-------

WE STI-- DON-- SEE THE NEE- FOR 31 CHA------ NAM--
IN THE GEN---- CAS-

YOU-- TRU--

Chu Moo

CHU-- MOO--
FOR-- INC-

P.S- MR. FRE- THO---- IS NOT AN EMP----- OF
FOR-- INC-

A STACK DIAGRAM UTILITY

Barry A. Cole
3450 Sawtelle Blvd. #332
Los Angeles, CA 90066

INTRODUCTION AND CONCEPT

A year and a half ago, when I was still fairly new to FORTH, I spent a lot of time drawing pictures of stacks as I made up programs. I crumpled them up and started over each time I changed them. As sections were debugged, I drew up another copy to document the code. When I found an error, I would have to redraw whole series of stacks, just as a cartoonist would have to change a whole series of frames. It soon became clear that I was expending time to do rather tedious work. I came up with an idea for an automated tool to update these diagrams. I thought up a way to represent the stack data easily and an approach to implement the tool. The original implementation was done in 8080 polyFORTH by my co-worker Greg Toussaint. We collaborated in the initial debugging and then passed it back and forth over the next four months. After nearly a year in active use, I converted it to fig-FORTH and updated several messy areas to be more straightforward. The results of these pursuits are detailed in this paper for more general consumption.

ORIGINAL IMPLEMENTATION

The original program was going to take push and pop information from the keyboard to generate pictures of what was on the stack. It became immediately clear that the stack could more easily be represented horizontally than down the page. We chose to put the stack to the right so that the size of the stack could be read like a bar graph. I figured that if I represented each item on the stack as an address pointing to a count and printable string, that many of the stack diagram words would be identical to the FORTH word equivalent. Thus, DUP, OVER, DROP as well as many other primitives would be coded before I started. Even as it was being

built, the tool grew to get the source codes directly from disk and then to generate a printer format spool file also onto FORTH screens. Keeping track of values when an IF was encountered and restoring them on ELSE and THEN was added. This generates a warning message if the two paths leave different numbers of parameters on the stack. Finally, concatenation of strings for algebraic and logical expressions was added.

USAGE AND OPERATION

The main routines called by a user are:

screen# DOC defname	to document 1 definition
screen# SDOC	to document a whole screen
screen# PRTDOC	to print from given screen
PDOC	to print last documentation

The program clears the display stack before each colon definition. A search is made for the first colon on SDOC or the specified name following a colon on DOC. The name of the function is displayed along with the currently empty stack contents. It requires user input to continue since the entry conditions of the routine are unknown. It prompts "DROP?" to see how many excess elements should be dropped from the stack, A carriage return suffices to leave it alone. It continues with the prompt , "PUSH VALUE?". For each symbolic name of a value on the stack, a free form name should be typed followed by a carriage return, The prompt will be repeated until a line consisting of only a carriage return is typed. There are no limitations imposed on the input, however, it is advised that nulls and tabs should not be included as this will detract from the clarity of the final output. The program will then continue reading words from the source screen and generating output lines to the console and spool file.

In a typical sequence, up to about a dozen lines will be handled without intervention. For example, occurences of DUP, DROP, and numeric literals will be processed automatically. When a @ is encountered, it will revert to the prompts since it is not known what a symbolically appropriate name is for the fetched value.

Processing will terminate with an "OK" for sucessful completion of the screen or colon for SDOC or PDOC, respectively. If stack underflow occures, it will abort. It is good practice to do a FORTH after an abort condition to insure that the stack vocabulary is properly exited. A user abort is also provided. This is accomplished by typing an escape key followed by a carriage return in response to the "PUSH VALUE?" prompt.

SAMPLE DIALOG

The package creates a special stack vocabulary as well as the user entry points. The use of the package is best seen by example. Figure 1 is a sample dialog. Notice how little intervention is required and how the ELSE restores the stack values. Figure 2 is the source that was used in the examples. Figure 3 is the printer output as displayed by PDOC.

FIGURE 1

```
100 SDOC
ANALYZE       |
DROP?
PUSH VALUE? addr
PUSH VALUE? len
PUSH VALUE?
ANALYZE       | addr  len
SWAP          | len  addr
INCH          |
DROP?
PUSH VALUE? char
PUSH VALUE?
INCH          | len  addr  char
DUP           | len  addr  char  char
7F            | len  addr  char  char  7F
-             | len  addr  char  (char-7F)
IF            | len  addr  char
DUP           | len  addr  char  char
0D            | len  addr  char  char  0D
-             | len  addr  char  (char-0D)
IF            | len  addr  char
DUP           | len  addr  char  char
OUCH          |
DROP? 1
PUSH VALUE?
```

FIGURE 1 (cont.)

```
OUCH          | len  addr  char
OVER          | len  addr  char  addr
C!            | len  addr
1+            | len  (addr+1)
SWAP          | (addr+1)  len
1-            | (addr+1)  (len-1)
ELSE          | len  addr  char
(             | len  addr  char
DROP          | len  addr
SWAP          | addr  len
DROP          | addr
20            | addr  20
OUCH          |
DROP? 1
PUSH VALUE?
OUCH          | addr
0             | addr  0
OVER          | addr  0  addr
C!            | addr
0             | addr  0
THEN          | addr  0
ELSE          | len  addr  char
(             | len  addr  char
DROP          | len  addr
8             | len  addr  8
OUCH          |
DROP? 1
PUSH VALUE?
OUCH          | len  addr
1-            | len  (addr-1)
SWAP          | (addr-1)  len
1+            | (addr-1)  (len+1)
THEN          | (addr-1)  (len+1)
;             | (addr-1)  (len+1)
OK
```

FIGURE 2

```
 OK
100 LIST
SCR # 100
  0
  1 : ANALYZE  SWAP INCH  DUP 7F - IF DUP 0D - IF
  2   DUP OUCH OVER C! 1+ SWAP 1-
  3   ELSE ( CR) DROP SWAP DROP 20 OUCH 0 OVER C! 0 THEN
  4   ELSE ( DELETE)  DROP 8 OUCH 1- SWAP 1+ THEN   ;
  5
  6
  OK
```

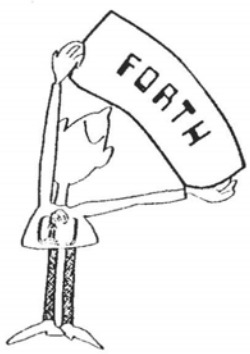


FIGURE 3

```
PDOC STACK DIAGRAM - SCREEN # 100

ANALYZE       | addr  len
SWAP          | len  addr
INCH          | len  addr  char
DUP           | len  addr  char  char
7F            | len  addr  char  char  7F
-             | len  addr  char  (char-7F)
IF            | len  addr  char
DUP           | len  addr  char  char
0D            | len  addr  char  char  0D
-             | len  addr  char  (char-0D)
IF            | len  addr  char
DUP           | len  addr  char  char
OUCH          | len  addr  char
OVER          | len  addr  char  addr
C!            | len  addr
1+            | len  (addr+1)
SWAP          | (addr+1)  len
1-            | (addr+1)  (len-1)
ELSE          | len  addr  char
(             | len  addr  char
DROP          | len  addr
SWAP          | addr  len
DROP          | addr
20            | addr  20
OUCH          | addr
0             | addr  0
OVER          | addr  0  addr
C!            | addr
0             | addr  0
THEN          | addr  0
ELSE          | len  addr  char
(             | len  addr  char
DROP          | len  addr
8             | len  addr  8
OUCH          | len  addr
1-            | len  (addr-1)
SWAP          | (addr-1)  len
1+            | (addr-1)  (len+1)
THEN          | (addr-1)  (len+1)
;             | (addr-1)  (len+1)
OK
```

CONDITIONALS

The IF...ELSE...THEN construct automatically saves and restores the stack values. A mismatch in number along the two paths produces a warning message,

"STK ERROR, ELSE -m THEN -n"

where m is the number of parameters left on the stack at the end of the IF clause and n is the number left when the THEN is encountered. The DROP/PUSH prompts are presented for the user to attempt recovery. A known cause of this message is a -DUP preceding the IF, as this is not handled.

SPOOLING TO DISK

To be useful, a hard copy of the output without all the intermediate operator conversation is useful. It is also quite possible that a machine readable version would be handy to facilitate distribution of the documentations. A spool file is generated to satisfy these requirements. It may later be displayed or printed by PDOC.

The spool file contains the encoded screen from which the diagram was made followed by variable length lines separated by carriage return characters. The file is terminated by an ascii null character. It resides on a set of consecutive screens. The first screen and maximum number are determined by literals in SPIT and PRTDOC. I use 10 screens starting at 230. These may be copied elsewhere and printed by PRTDOC. Failure to copy them will cause the listing to be lost the next time a function or screen is diagrammed.

IMPLEMENTATION PROBLEMS AND SOLUTIONS

It is important not to search the standard vocabularies when diagramming stacks. This is because actions are different for the same name, depending upon state. By way of example, for the operator + must concatenate the symbolic name strings representing these elements with an embedded plus sign, rather than adding the top two elements on the stack. Also, not all operators are defined. On detection of this case, the diagrammer must shift control to the operator prompt section. In polyFORTH, this was accomplished by defining a new vocabulary and having it be the only one searched. in fig-FORTH, this option is not directly available since the FORTH vocabulary is searched after the current vocabulary. This may be solved by carefully breaking links with zero entries, or alternatively by defining a special dictionary search routine that stops at some fence value. I chose the latter.

It wasn't obvious until the implementation began that operators would require concatenation of their identifying strings. It was also decided that parenthesis would be placed around each level of expression nesting so that ambiguity could be eliminated without rearranging expressions for precedence. This occasionally leads to expression such as ((array+2)+2). This is unavoidable since even the constants within the expression are treated as strings rather than numbers. Thus, the example cannot be reduced to (array+4).

Error recovery is not nearly as good as I'd like it to be. Stack underflow in the diagramming session is generally fatal. Due to the amount of bookkeeping already being done, there is no provision for retracting answers after wrong data has been put on the diagram stack. This is inconvenient in a first pass through a function, but has not proved to be a problem once a feel for the tool and the function being diagrammed has been acquired.

Provision is left for user defined functions in the last two screens of the diagram source. This allows commonly used functions to be handled in an automated fashion. This makes it very easy to define composite functions such as, 1- as the sum of its component parts. For outside of functions, constant and variable

have been redefined to put their own name on the stack. Before this facility was added, I always retyped the variable name manually when it came up.

The spool function and some of the source reading routines such DOC assume that screen blocks are contiguous 1,024 byte areas. Those functions using BLOCK will have to be rewritten if this is not the case in your system. I recommend that you instead generate a new system with 1k buffers as that is faster and more flexible.

WEAKNESSES AND PROPOSED FUTURE EXTENSIONS

The diagrammer presently does not keep track of the contents of the return stack. This requires uses of R> and I go to the operator for clarification. Try a pencil for now. This could be added in a similar fashion as IF..ELSE..THEN by an additional stack.

The area of error recovery is ripe for suggestions. Perhaps some dummy buffer area could be added and tested in PSTAK. This would allow detection prior to destruction on stack underflow. Backing up by reading backwards would be nice but also very difficult to implement.

CONCLUSION

Now that the tool has been built, its real function is more evident. It is still used for documenting words as originally intended, however, its primary usage is debugging and validating code. It has also proved to be very useful as a teaching aid to explain what is going on within the stack. I hope it will be as useful to you as it has been to me.

A STACK DIAGRAM GLOSSARY

VARIABLES

:BK The base block number for spooled stack diagram.

:LN Line number being printed. Used for page headings.

:SC Current screen number being spooled.

IFPTR The address of top of IF stack. Used to restore values on stack for IF...ELSE...THEN construct.

IFST The area reserved for pointers to previous stack contents. It is used to restore the stack on ELSE and THEN clauses.

SPL A temporary variable used by :NFD to retreat the spool file to erase the unknown stack prior to operator specification of what is added or dropped.

SPOOL Offset into spooled print file.

SUM The sum of differences in two strings. Used in -TEXT. Value is 0 for a text match and nonzero if different.

T1 Pointer to current input word in memory (type format).

CONSTANTS

LLIMIT The limit address for dictionary search to keep from using standard FORTH words from within the STACK diagrammed words.

FUNCTIONS

--- 'FIND pfa length true (found)

--- 'FIND false (not found)

This is the same as -FIND except that the true condition is set only if the work is found above LLIMIT. This restricts the search to stack vocabulary words.

--- ('('

A string constant used for building expressions when arithmetic or logical operations are encountered in the diagrammed input string.

Defined back to its original state after being used as a concatenation token, this marks the beginning of a comment.

All text following it is ignored until the next) .

Tests two strings for not equal.

---) ')'

A string constant used for building expressions when arithmetic or logical operations are encountered in the diagrammed input string.

st1 st2 -TEXT cond

True if the two strings differ.

val 1- val-1

Decrements the top of stack value by one.

v1 v2 2DROP ---

Drops the top 2 elements off the stack and discards them.

--- :: ---

This is the stack diagram redefinition of colon. It diagrams the word following it instead of compiling.

It is invoked by colon as the very last definition from within this package.

st1 st2 :C st3

Concatenates two strings into a single combined string. It is used to build expressions when operators are encountered in the screen to be diagrammed.

--- :HEAD ---

Prints the header for a line of output to the console and also the spool file.

--- :KILL ---

Removes and discards the top of the IFST.

--- :NFD addr

This is called when the word being analyzed is not in the special stack vocabulary. It checks for valid numbers. If this test is passed, it returns a pointer to that string. Otherwise, it invokes SKBD to get user help.

adr :PSH ---

Pushes the address of a level of the stack values onto the separate IF stack. This is used for IF..ELSE..THEN stack restoration and checking.

adr :RST e11 e12 etc

Restores the stack from the IFST stack. Does not affect the IFST.

adr :SAV ---

Saves current stack element list on the IFST. Does not affect the parameter stack.

--- :SP ---

Marks the end of the spool file with a zero.

adr ?NUM cond

Checks current word to determine whether it qualifies as a legal hexadecimal number.

--- CONSTANT ---

A defining word which causes the name of the defined word to be put on the stack when that word is encountered.

--- DEPTH depth

Computes the depth of the stack in items.

scr# DOC ---

Searches for a colon followed by the word whose name follows this invocation on the specified screen. It aborts if the definition is not on the specified screen. Otherwise, it commences to generate the diagram for the word specified.

--- ELSE e11 e12 etc

Clears the stack and then restores it from IFST.

--- ESC ---

Aborts the package if an escape key was the first key pressed in answer to the "PUSH?" prompt. The vocabulary reverts to FORTH; however, the stack diagram package is still loaded and ready to go.

--- G-HERE adr cond

Moves a string from PAD into the dictionary. It allots the space and leaves the address of the item and a true cond if successful. It leaves only a false cond if no valid string was found.

expr G(1) op(expr)

Builds an expression from a simpler expression. At execution time of the following word, the top of the stack is enclosed in parenthesis and preceded by the operation symbol. It is used for unary operations. eg. -(name)

expr G(2) op(expr)

Similar to G(1) except that unary operation is also enclosed within the parenthesis. eg. (name*2)

espr G(3) op(expr)

Similar to G(1) except that binary operation is also enclosed within the parenthesis. eg. (val1+val2)

inadr GBLD ---

An auxiliary word used to build a named string in the dictionary from the word following GBLD. This is used at compile time of the stack diagram package.

--- GWRD ---

A defining word for building strings into the dictionary at compile time of the stack package. On invocation of the new word, the address of the string displaying its own name is put on the stack. The word that follows GWRD is read twice at compile time, once for the name of the function, and a second time to be placed in string format into the dictionary. This is used to build up constant words for the diagramming package.

cond IF ---

Drops the condition flag from the top of the stack without evaluating it. It then invokes :SAV for ELSE restoration and THEN error checking.

adr cnt MTYP

Types the message to the screen and also passes the parameters to STYP for spooling.

src dst len MVB dst len src+1 src

Intermediate function to set up for MVDEL.

src dst delim MVDEL adr

Move a string from the source to the destination address until the specified delimiter is encountered. This is used to build data strings within the dictionary.

--- PDOC ---

Prints the latest generated diagram from the default spool file blocks.

--- PHDG ---

Prints the top of page heading and sets the lines per page count. Used by PRTDOC.

blk# PRTDOC ---

Prints the stack diagram from the spool file whose starting block is the specified blk#.

--- PSTAK ---

Prints all words from the string addresses on the stack. The top element is printed to the right of previous elements. The stack is unchanged.

adr cnt PWRD ---

Prints one word via MTYP. Used by PSTAK.

--- REPEAT ---

Functionally identical to the re-defined THEN.

scr# SCRST scr#

Resets the spool pointer and places the screen number into the beginning of the output spool to be used in top of page headers by PDOC.

scr# SDOC ---

Documents one whole screen by executing it, using the diagram definitions.

--- SKBD ---

This scans the keyboard for user interaction. It generates the "DROP?" and "PUSH VALUE?" prompts. It is invoked whenever intervention is required in the diagramming process.

char SPIT ---

Writes character out to disk spool file.

--- STACK ---

This is the name of the vocabulary containing this package.

adr cnt STYP ---

Similar to TYPE but spools to disk rather than typing to the screen. Outputs an additional two blanks after the message.

adr T; cond

Tests the current string for a match to the FORTH word semicolon. This is used to exit DOC.

--- THEN ---

Re-defined in the stack vocabulary, this cleans up the IFST. If the depth of the stack has changed from before the ELSE, it issues a warning and calls SKBD to allow the user to correct a stack depth disparrity between the IF and ELSE clauses.

--- VARIABLE ---

A defining word which causes the name of the defined word to be put on the stack when that word is encountered.

```
102 THIAD

SCR # 102
  0 ( stack diagram package   1 of 14   B. A. Cole  810326)
  1
  2 : 1-      1 - ;
  3 : 2DROP   DROP DROP ;   -->
  4
  5   Calling sequences:
  6     screen DOC defname
  7     screen SDOC
  8            PDOC
  9
 10 DOC builds stack diagram for one definition.
 11 SDOC builds stack diagrams for entire screen.
 12 PDOC prints stack diagram built by DOC or SDOC.
 13
 14
 15


SCR # 103
  0 ( stack diagram package   2 of 14   B. A. Cole  810326)
  1 0 VARIABLE SPOOL              ( offset into spooled print file )
  2 0 VARIABLE SPL                (                                )
  3 0 VARIABLE T1                 ( pointer to current input word  )
  4 0 VARIABLE SUM                ( in -TEXT : true for difference )
  5 0 VARIABLE :LN                ( line number being printed      )
  6 0 VARIABLE :BK                ( base block# for spooled diagram)
  7 0 VARIABLE :SC                ( current screen # spooled       )
  8 0 VARIABLE IFPTR              ( address of top of IF stack     )
  9 0 VARIABLE IFST  20 ALLOT     ( define IF stack                )
 10    IFST IFPTR !               (   and initialize to empty      )
 11 -->
 12
 13
 14
 15


SCR # 104
  0 ( stack diagram package   3 of 14   B. A. Cole  810326)
  1 : SPIT SPOOL @ 1024 /MOD  9 MIN 230 +
  2        BLOCK + C! 1  SPOOL +! UPDATE ; ( spool one character  )
  3
  4 : :SP SPOOL @ 0 SPIT SPOOL ! ;         ( mark spool file end  )
  5
  6 : DEPTH  S0 @ SP@ - 2 / 1 - ;          ( compute stack depth  )
  7
  8 : STYP SWAP DUP ROT + SWAP DO          ( spool word           )
  9        I @ SPIT LOOP 32 SPIT 32 SPIT ;
 10
 11 : MTYP 2DUP STYP TYPE 2 SPACES :SP ;   ( type and spool word  )
 12
 13 : PWRD DUP C@ 72 >
 14        IF   ." ERR "  ABORT            ( print encoded word   )
 15        ELSE COUNT MTYP THEN ; -->

***** from:  Barry A. Cole  Los Angeles, CA  213-390-3851 *****
OK
```

```
105 TRIAD

SCR # 105
  0 ( stack diagram package   4 of 14   B. A. Cole  810326)
  1 : PSTAK DEPTH IF SP@ 2 - S0 @ 2 -  ( print all words on stack )
  2   BEGIN DUP @ PWRD 2 - 2DUP = END
  3   DROP DROP THEN ;
  4
  5 HEX    : T;  # 3B01 - ;     DECIMAL  ( test for not semicolon    )
  6
  7 : :C SWAP COUNT DUP ROT SWAP
  8   HERE 1+ SWAP CMOVE DUP HERE 1+ +
  9   ROT COUNT ROT SWAP DUP >R CMOVE
 10   R> + DUP HERE DUP ROT SWAP
 11   C! SWAP 1+ ALLOT ;                ( concatenate 2 words )
 12
 13 -->
 14
 15


SCR # 106
  0 ( stack diagram package   5 of 14   B. A. Cole  810326)
  1 : ?NUM DUP 1+ C@ 45 = +                ( test if input is number)
  2   BEGIN 1+ DUP C@ 16 DIGIT
  3   WHILE DROP REPEAT  C@ 32 = ;
  4 : :HEAD CR 13 SPIT 10 SPIT DUP         ( print line header)
  5   COUNT 10 MIN SWAP OVER MTYP 11
  6   SWAP - 0 DO 32 SPIT SPACE LOOP
  7   ." | " 124 SPIT  32 SPIT :SP ;
  8 : MVB   ROT DUP 1+ SWAP ;              ( setup for MVDEL)
  9 : MVDEL BEGIN MVB C@ >R MVB I SWAP C! ( src dst len --       )
 10   ROT DUP R> + END ROT  2DROP ;       ( move s to d til delim)
 11 : G-HERE HERE PAD HERE 1+ 0 MVDEL     ( string into memory)
 12   HERE - 1- DUP 1- DUP ROT SWAP DUP
 13   IF   HERE C! ALLOT
 14   ELSE 2DROP 2DROP 0 THEN ;
 15 : ESC PAD C@ 27 = IF ." ESC " ABORT THEN ;  ( escape )  -->


SCR # 107
  0 ( stack diagram package   6 of 14   B. A. Cole  810326)
  1 VOCABULARY STACK   0 CONSTANT LLIMIT ( filled in later)
  2
  3 : SKBD CR ." DROP? " KEY DUP EMIT   ( scan kbd for drop,pushes)
  4   48 XOR DUP 9 <
  5   IF   DEPTH 1 - MIN 0 DO DROP LOOP
  6   ELSE DROP THEN
  7   BEGIN CR ." PUSH VALUE? " PAD 80
  8     EXPECT ESC G-HERE 0= END ;
  9
 10 : :NFD  T1 @ ?NUM                     ( handle word not found)
 11   IF   T1 @                           ( number)
 12   ELSE SKBD SPOOL @ SPL !             ( undefined)
 13        T1 @ :HEAD  DROP SPL @ SPOOL ! THEN ;
 14   -->
 15

***** from:  Barry A. Cole  Los Angeles, CA  213-390-3851 *****
OK



108 TRIAD

SCR # 108
  0 ( stack diagram package   7 of 14   B. A. Cole  810326)
  1 : 'FIND  -FIND DUP IF 2DROP DUP LLIMIT >
  2         IF 1 ELSE 0= THEN  THEN ;
  3
  4 : ::                                   ( stack diagram redef of :)
  5   BEGIN 'FIND HERE DUP DUP T1 !        ( word to mem and printed)
  6         C@ 1+ ALLOT :HEAD T;
  7   WHILE IF CFA EXECUTE ELSE :NFD THEN  ( execute each word and)
  8         ?STACK  PSTAK  REPEAT          (   print stack)
  9   DROP PSTAK CR 13 SPIT 10 SPIT :SP SP! ;
 10
 11 : SCRST 0 SPOOL ! DUP 256 /MOD        ( reset spool ptr and place)
 12         SWAP SPIT SPIT ;              (   screen# in spool)
 13
 14 : SDOC  SCRST STACK LOAD              ( document 1 screen)
 15         [COMPILE] FORTH ;  -->


SCR # 109
  0 ( stack diagram package   8 of 14   B. A. Cole  810326)
  1 : PHDG 12 EMIT                    ( print heading on top of page )
  2   ." STACK DIAGRAM - SCREEN # "
  3   :SC ? CR CR 54 :LN ! ;
  4
  5 : PRTDOC  DUP :BK !  BLOCK #  ( print diagram from spec scr)
  6   :SC ! PHDG  10240 2 DO
  7     I 1024 /MOD :BK @ + BLOCK + C@ -DUP
  8     IF   DUP EMIT 10 =
  9          IF :LN @ 1 - DUP :LN ! 0=
 10            IF PHDG THEN THEN
 11     ELSE LEAVE THEN LOOP ;
 12
 13 : PDOC 230 PRTDOC ;              ( print last generated diagram)
 14
 15 -->
```

```
SCR # 110
  0 ( stack diagram package   9 of 14   B. A. Cole  810326)
  1 : -TEXT 0 SUM ! SWAP 0                 ( true if 2 strings differ)
  2   DO OVER I + C@ OVER I + C@ - SUM +! LOOP
  3   2DROP SUM @ ;
  4
  5 : WTEST OVER DUP C@ 1+ SWAP -TEXT ;      ( test 2 words for <>)
  6
  7 : DOC SCRST HERE 32 WORD DUP C@ 1+ ALLOT ( document 1 def)
  8   BLK @ >R IN @ >R 0 IN !  SWAP BLK !
  9   BEGIN  58 WORD  IN @ >R  32 WORD       (     find colon,word)
 10   HERE WTEST
 11   WHILE R> 1024 +
 12     IF ." NOT FOUND" ABORT THEN REPEAT
 13   DROP   R> IN !   STACK ::             (     now go from here)
 14   [COMPILE] FORTH R> IN ! R> BLK ! ;
 15 -->

***** from:  Barry A. Cole  Los Angeles, CA  213-390-3851 *****
OK



111 TRIAD

SCR # 111
  0 ( stack diagram package  10 of 14   B. A. Cole  810425)
  1 : :PSH   2 IFPTR +! IFPTR @ ! ;
  2 : :SAV HERE :PSH SP@ DEPTH 1 - DUP ,
  3   DUP + HERE SWAP DUP ALLOT CMOVE ;
  4 : :RST IFPTR @ @ DUP @ DUP 2 * ROT + SWAP -DUP IF
  5       0 DO DUP @ SWAP 2 - LOOP THEN  DROP ;
  6 : :KILL  -2 IFPTR +! ;
  7   STACK DEFINITIONS     HERE ' LLIMIT !
  8
  9 -->
 10
 11
 12
 13
 14
 15


SCR # 112
  0 ( stack diagram package  11 of 14   B. A. Cole  810425)
  1
  2 : THEN DEPTH IFPTR @ @ 1+ C@ -DUP
  3   IF 128 - 2DUP -
  4         IF   CR ." STK ERROR, ELSE -" .
  5              ."  THEN -" . SKBD
  6         ELSE 2DROP THEN
  7   ELSE DROP THEN :KILL ;
  8
  9 : ELSE DEPTH 128 + >R SP! :RST   R> IFPTR @ @ 1+ C! ;
 10
 11 : IF DROP :SAV ;   FORTH DEFINITIONS  -->
 12
 13
 14
 15


SCR # 113
  0 ( stack diagram package  12 of 14   B. A. Cole  810326)
  1 : GBLD  IN ! 32 WORD HERE C@ 1+ ALLOT ;
  2 : GWRD  IN @ <BUILDS GBLD
  3         DOES>  ;
  4   GWRD )      GWRD (
  5 : G(1)  IN @ <BUILDS GBLD
  6         DOES>  SWAP )             ( SWAP :C :C ;
  7 : G(2)  IN @ <BUILDS GBLD
  8         DOES>        ) :C :C     ( SWAP :C ;
  9 : G(3)  IN @ <BUILDS GBLD
 10         DOES>  SWAP ) :C :C :C ( SWAP :C ;
 11 : ( 41 WORD ; IMMEDIATE              -->
 12
 13
 14
 15

***** from:  Barry A. Cole  Los Angeles, CA  213-390-3851 *****
OK
```

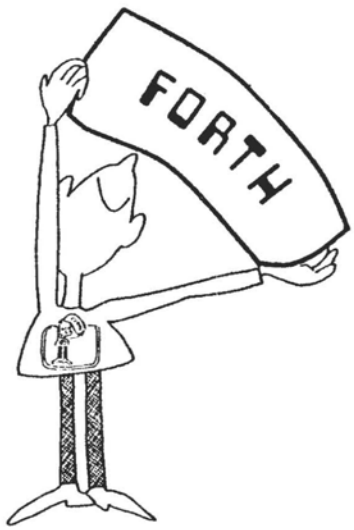
```
114 THIAD

SCR # 114
  0 ( stack diagram package  13 of 14   B. A. Cole  810425)
  1
  2   STACK DEFINITIONS
  3 :    32 WORD  HERE DUP  C@ 1+ ALLOT  ;
  4 : DUP DUP ;       : ROT ROT ;     : SWAP SWAP ; : OVER OVER ;
  5 : R>  DROP ;      : DROP DROP ;   : . DROP ; : HEX HEX ; : MSG ;
  6 : DECIMAL DECIMAL ; : 2DUP 2DUP ; : ! DROP DROP ; : C! ! ;
  7 : +! ! ;          : , DROP ;      : SPACE ;       : [ ;    : ] ;
  8 : 2! ! DROP ;     : ? DROP ; : ALLOT DROP ; : BLANK ! ;
  9 : C, DROP ;       : DO ! ; : DUMP ! ; : EMIT DROP ;  : -DUP DUP ;
 10 : END DROP ;      : ERASE ! ; : +LOOP DROP ; : /LOOP DROP ;
 11 : EXPECT ! ;      : MOVE ! DROP ;    : LEAVE ;
 12 : SPACES DROP ; : TYPE ! ; : BEGIN ; : LOOP ; : \ ;   : CR ;
 13 : ENDIF  THEN ; : REPEAT THEN ;    : WHILE IF ;
 14 : AGAIN ;        : UNTIL  ! ;
 15 -->

SCR # 115
  0 ( stack diagram package  14 of 14   B. A. Cole  810425)
  1 GWRD 0            GWRD 1            GWRD 2            GWRD 3
  2 G(1) -   : MINUS - ;      G(3) -   G(3) +      G(3) *   G(3) /
  3 G(3) &   : AND & ;        G(3) |   : OR | ;     G(3) %   : XOR % ;
  4 G(2) +1 : 1+ +1 ;         G(2) -1 : 1- -1 ;     G(2) +2 : 2+ +2 ;
  5 G(2) *2 : 2* *2 ;         G(2) /2 : 2/ /2 ;     G(2) -2 : 2- -2 ;
  6 G(1) NOT                  GWRD I                GWRD cond
  7 GWRD here    : HERE here ;  GWRD pad            : PAD pad ;
  8 : = 2DROP cond ;            : ALLOT DROP ;
  9 : VARIABLE DROP GWRD ;      : CONSTANT VARIABLE ;
 10 : < = ;                     : > = ;
 11 : ;S QUIT ;  : --> QUIT ;  : ." 34 WORD ;        : : :: ;
 12 FORTH DEFINITIONS    ;S
 13
 14
 15

SCR # 116
  0
  1
  2
  3
  4
  5
  6
  7
  8
  9
 10
 11
 12
 13
 14
 15

***** from:  Barry A. Cole  Los Angeles, CA  213-390-3851 *****
OK
```



FORTH CLASS

Date: June 22 - 26

Where: Humbolt State University
Arcata, CA 95521

Who: Kim Harris and Henry Laxen

What: Intensive 5-day course on the use of FORTH

Cost: $100 - $140 plus room and board

How: Call Prof. Ron Zammit
(707) 826-3275

MMS-FORTH FOR STRINGY FLOPPIES

Kalth Microsystems will make available to all licensed MMS-FORTH users a modified version that runs on the TRS-80 with an EXATRON stringy floppy. This modification is said to make MMS-FORTH operate as it would on a disk except for the speed. Users retain the capability to switch back to cassette operation with a single command. Implementation includes the normal read/write block commands plus a number of new utility words. The modification is available on ESF wafer for $14.95 including shipping. For more information contact:

Kalman Fejes
KALTH MICROSYSTEMS
P.O. Box 5457, Station F
Ottawa, Ontario K2C 3J1
Canada

How to form a FIG Chapter:

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

2. Send to FIG in San Carlos, CA 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 a.m.

Southern California

Los Angeles

4th Sat	FIG Meeting, 11:00 a.m., Allstate Savings, 8800 So. Sepulveda, L.A. Call Phillip 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. Call Guy Kelly at (714) 268-3100, x 4784 for site.

Massachusetts

3rd Wed	MMSFORTH Users Group, 7:00 p.m., Cochituate, MA. Call Dick Miller at (617) 653-6136 for site.
Seattle	Chuck Pliske or Dwight Vandenburg at (206) 542-7611.
Potomac	Paul van der Eijk at (703) 354-7443 or Joel Shprentz at (703) 437-9218.
Tulsa	Art Gorski at (918) 743-0113.
Texas	Jeff Lewis at (713) 719-3320 or John Earls at (214) 661-2928 or Dwayne Gustaus at (817) 387-6976. John Hastings (512) 835-1918.
Phoenix	Peter Bates at (602) 996-8398.
New York	Tom Jung at (212) 746-4062.
Detroit	Dean Vieau at (313) 493-5105.
England	FORTH Interest Group, c/o 38, Worsley Road, Frimley, Camberley, Surrey, GU16 5AU, England
Japan	Mr. Okada, Presdient, ASR Corp. Int'l, 3-15-8, Nishi-Shimbashi Manato-ku, Tokyo, Japan.
Canada Quebec	Gilles Paillard at (418) 871-1960.
West Germany	Wolf Gervert, Roter Hahn 29, D-2 Hamburg 72, West Germany,(040) 644-3985.

Publishers Note:

Please send notes (and reports) about your meetings.

FORTH INTEREST GROUP
MAIL ORDER

		USA	FOREIGN AIR
___	Membership in FORTH INTEREST GROUP and Volume III (6 issues) of FORTH Dimensions. **Check one: ___NEW or ___RENEWAL**	$12	$24
___	Volume II of FORTH DIMENSIONS (6 issues)	$12	$15
___	Volume I of FORTH DIMENSIONS (6 issues)	$10	$14
___	fig-FORTH Installation Manual, containing the language model of fig-FORTH, a complete glossary, memory map and installation instructions	$10	$14
___	Assembly Language Source Listings of fig-FORTH for specific CPU's and machines. The above manual is required for installation. Check appropriate box(es. **Price per each.**		
	___ 1802 ___ 6502 ___ 6800 ___ 6809 ___ 8080 ___ 8086/8088 ___ 9900 ___ APPLE II ___ PACE ___ ALPHA MICRO ___ PDP-11 ___ NOVA	$10	$14
___	PROCEEDINGS 1980 FORML (FORTH Modification Lab) Conference	$25	$35
___	FORTH-79 Standard, a publication of the FORTH Standards Team	$10	$13
___	Using FORTH, by FORTH, Inc. This is the best users manual.	$25	$32
___	Kitt Peak Primer, by Stevens. An indepth self-study primer.	$25	$35
___	BYTE Magazine Reprints of FORTH articles, 8/80 to 4/81	$ 5	$ 8
___	FIG T-shirts: ___ Small ___ Medium ___ Large ___ X-Large	$10	$12
___	Poster/1981 Calendar, Aug 1980 BYTE cover, 18 x 22"	$ 5	$ 8
___	FORTH Programmer's Reference Card. If ordered separately, send a stamped, addressed envelope.		FREE
	TOTAL	$_____	

NAME_____MAIL STOP/APT_____

ORGANIZATION_____(If company address)

ADDRESS_____

CITY_____STATE_____ZIP_____COUNTRY_____

VISA #_____MASTER CHARGE #_____

(Minimum of $10.00 on charge cards)

Make check or money order in US Funds, payable to: **FIG**. All prices include postage. ***No purchase orders***

FORTH INTEREST GROUP PO BOX 1105 SAN CARLOS, CA 94070

FORTH CLASSES

June 22–26

Humboldt State

Eureka,CA

See Page 32

June 23 & July 24

College of Notre Dame

Belmont, CA

See Page 4

LEARN FORTH!

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

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

TIME DATED MATERIAL
DELIVER BEFORE JUNE 17

Address Correction Requested