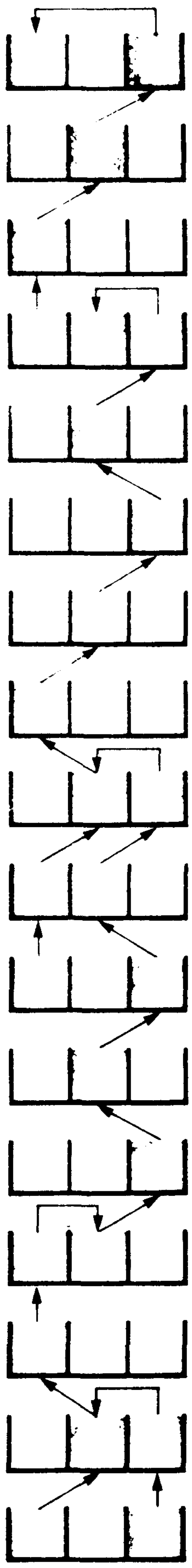


FORTH DIMENSIONS

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

Volume II
Number 6
Price $2.00

INSIDE

FORTH DIMENSIONS

Published by Forth Interest Group

Volume II No 6 March April 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 overseas 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 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 2,800 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 theme of this month's FORTH DIMENSION is practical applications.

During the last two years or so I have heard from many FIG members who seem to have a common problem— "Now that I have FORTH, where do I go from here?" In addition many of us seem to be re-inventing code that others have already running, just because we are unaware of its existence.

In short, FIG members are suffering from a common problem—failure to communicate. Fortunately, this is an easily cured problem. FORTH DIMENSIONS is our communications vehicle, all we have to do is use it.

The mechanics are simple. FORTH DIMENSIONS is seeking short universal tool type code segments for publication. If you have some code that you have found especially useful and can explain its function and use, please contact the editor at FORTH DIMENSIONS.

YOU DON'T HAVE TO BE A WRITER! You will be sent a publication kit that leads you through the writing process. You will also be given all the help necessary by the FORTH DIMENSIONS editorial staff.

FIG members already have a reputation as creative problem solvers, now if we will just share and exchange our ideas, the permutations of that process boggle the mind. I am looking forward to enthusiastic response to this new approach that will benefit all.

C. J. Street

PUBLISHER'S COLUMN

It's the end of the FIG year and renewals are piling in. (Have you renewed?). Some of our newer members might be confused about renewing. If you recently joined FIG and received back issues of Volume II of FORTH DIMENSIONS then it is time to renew for Volume III and your March 1981 to March 1982 membership.

A number of other items of interest:

- FIG now has over 2800 members, worldwide
- FIG will have booths at the Computer Faire, April 3-5 in San Francisco and at the Jersey Computer Show in Trenton on April 25.
- There are a number of new listings — see order form at back
- Several reports from new chapters — lets see more
- Proceeding of 1980 FORML Conference is now available — see order form
- Looks like this is going to be our biggest year

Roy Martens

FORGIVING FORGET

Dave Kilbridge

Acknowledgment

I want to describe a FORTH system word which has come to be known as "smart FORGET" or even "Dave Kilbridge's smart FORGET." But the ideas involved appear in the State University of Utrecht, The Netherlands' FORTH system at least as early as 23 May 1978. The code presented here is a straightforward adaptation to the FIG model.

The Problem

The principal function of FORGET is to reclaim memory by locating in the dictionary the next word in the input stream and resetting the dictionary pointer (DP) to the beginning of the definition of that word. To avoid destroying vital parts of the system, no FORGETting is allowed below the address stored in FENCE. In the "dumb FORGET" of the original FIG model (see Screen 72), this address check is made on line 8.

But merely truncating the dictionary, even at a safe place, is not enough. The dictionary has a linked-list structure which allows it to be searched. If a link is left pointing into the "never-never-land" beyond the new value of DP, then the system may crash the next time a dictionary search uses that link.

These links are of two types: (1) VOCABULARY words have a link to the latest word in the vocabulary they name. "Dumb FORGET" adjusts this link (line 9) to point to the latest word which you don't FORGET, but only for the CURRENT and CONTEXT vocabularies. (Line 7 verifies that these are the same; this test was thought to give some extra protection against crashing. Any vocabulary not in CURRENT or CONTEXT may be trashed. (2) CURRENT and CONTEXT themselves point to vocabularies. If you FORGET the name of the CURRENT vocabulary, or any word before it in the dictionary, you may crash.

The Solution

"Smart FORGET" overcomes these hazards so effectively that I have never crashed by doing a FORGET. This is made possible by linking all the VOCABULARY words in the system into another linked list, enabling them to be located. The head of the list is stored in VOC-LINK. See the figure for the various fields in a VOCABULARY word.

How It Works

Refer to the code on Screen 18. On line 7, the name-field-address of the next input word is located in the dictionary; this is the point at which the dictionary will be cut off. An error message issues if this address is below the contents of FENCE. This cutoff address is saved on the return stack, and the head of the vocabulary list is put on the parameter stack. Now everything is ready for the real work.

The BEGIN ... WHILE ... REPEAT loop on lines 9-10 runs through all VOCABULARY words above the cutoff address and unlinks each from the list. If any such vocabularies are found, both CONTEXT and CURRENT are pointed to FORTH. This removes any links described as type (2) above.

Now the outer BEGIN ... UNTIL loop on lines 11-13 runs through the remaining VOCABULARY words. For each such word, the loop on line 12 finds the highest word below the cutoff address in the corresponding vocabulary. The vocabulary head is then pointed to this word, thus fixing the links of type (1) above.

Finally, DP is reset to point to the cutoff address (line 14).

Improvements

Executing FORTH DEFINITIONS if any VOCABULARY word is found beyond the cutoff address is unnecessarily drastic. One could test CURRENT and CONTEXT and only change them if they point beyond the cutoff, but it's probably not worth the trouble.

Extensions

1. In systems which allow dynamic chaining of vocabularies, one must check whether a vocabulary chained to is beyond the cutoff address. If so, it is replaced by FORTH. (The Utrecht system does exactly that.)

2. In later versions of the author's PACE system, a base-page pointer is allocated for each new defining word. These are released by FORGET. This is done by comparing pointer values with the cutoff address and does not involve the vocabulary structure.

```
SCR # 72
  0 (  ',  FORGET,                                      WFR-79APR28 )
  1 HEX    3   WIDTH   !
  2 : '          ( FIND NEXT WORDS PFA; COMPILE IT, IF COMPILING *)
  3     -FIND  0=  0  ?ERROR  DROP  [COMPILE]  LITERAL  ;
  4                                       IMMEDIATE
  5
  6 : FORGET              ( FOLLOWING WORD FROM CURRENT VOCABULARY *)
  7     CURRENT  @  CONTEXT  @  -  18  ?ERROR
  8     [COMPILE]  '  DUP  FENCE  @  <  15  ?ERROR
  9     DUP  NFA  DP  !  LFA  @  CURRENT  @  !  ;
 10
 11
 12
 13  -->
 14
 15
```

```
SCR # 18
  0 (  Smart FORGET                                 DJK-WFR-79DEC02 )
  1 : '          ( FIND NEXT WORDS PFA; COMPILE IT, IF COMPILING *)
  2     -FIND  0=  0  ?ERROR  DROP  [COMPILE]  LITERAL  ;
  3                                       IMMEDIATE
  4 HEX
  5
  6 : FORGET                      ( Dave Kilbridge's Smart Forget )
  7     [COMPILE]  '  NFA   DUP  FENCE  @ u<  15  ?ERROR
  8     >R  VOC-LINK  @  ( start with latest vocabulary )
  9  BEGIN  R  OVER  u<  WHILE   [COMPILE]  FORTH  DEFINITIONS
 10      @  REPEAT  DUP  VOC-LINK  !  ( unlink from voc list )
 11  BEGIN  DUP  4  -       ( start with phantom nfa )
 12      BEGIN  PFA  LFA  @  DUP  R  u<  UNTIL
 13      OVER  2  -  !  @  -DUP  0=  UNTIL  ( end of list ? )
 14    R>  DP  !  ;    -->
 15 This replaces Screen 72 of the F.I.G. Model.
```

SOME NEW EDITOR EXTENSIONS

Kim Harris

This article shows how to add two new commands to the FORTH editor which permit the replacement or insertion of multiple lines of a screen. This is a mini-application which demonstrates string input and output, adding new commands to the Forth editor, manipulating vocabularies, and a "terminal input processor" which prompts for input then processes it. Several variations in implementation are shown to illustrate different styles and refinements. If you are only interested in the final result, you can type in Screen 45 (in this article) into any standard fig-FORTH system which already has the FIG line editor (from screens 87 to 91 in the Installation Manual).

The use of the new commands will be illustrated by an example. Input is underlined; output is not. The symbol (CR) means to push the Carriage Return key (or equivalent).

To begin any editing of screen 100 you say

```
 100 LIST  EDITOR     (CR)
0   ( TEST SCREEN )
1   old 1st line
2   old 2nd line
3   old 3rd line
       . . .
```

To replace one or more lines starting at line 2, say

```
 2 NEW (CR)
0   ( TEST SCREEN )
1   old 1st line
2   -
```

The cursor is at the start of line 2 and waiting for you to enter new text. If you enter some text and a (CR), it will prompt you for a new line 3 and so on. This continues until you replace line 15 or enter only a (CR) at the start of a line. Then that line and any remaining ones are listed unchanged.

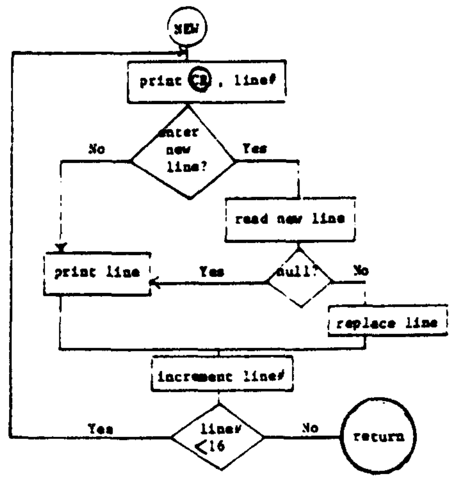
```
 2 NEW (CR)
0   ( TEST SCREEN )
1   old 1st line
2   new text for line 2 (CR)
3   something for line 3 (CR)
4  (CR) old 4th line
5   old 5th line
       . . .
```

A similar command UNDER lets you insert one or more lines starting at a specified line number.

```
 2 UNDER (CR)
0   ( TEST SCREEN )
1   old 1st line
2   new text for line 2
3   inserted line (CR)
4   another inserted line (CR)
5  (CR) something for line 3
6   old 4th line
7   old 5th line
       . . .
```

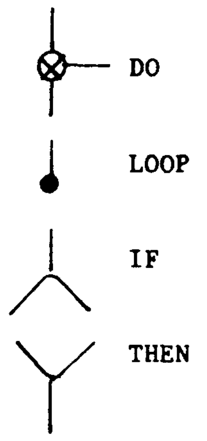
Any lines pushed off line 15 are lost.

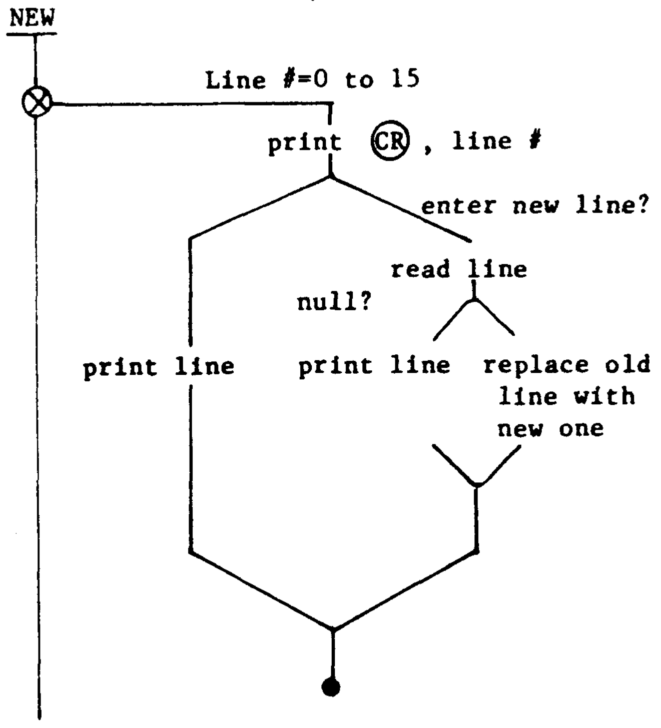
Let's design this application starting from the top. First consider the control flow for NEW and draw a flowchart. The one below is a traditional ANSI standard one.



This flow chart is poor. It is unstructured (i.e., "print line" is improperly shared by two IF structures), the loop structure requires two boxes which can be performed by the single word DO, and no symbol exists for the word LOOP. To program this flowchart, you either have to cheat or change the flowchart. An example of cheating is in Screen 12. This implementation of NEW is by Bill Ragsdale and works fine. The tricks are the words inside square brackets on lines 6 and 8. These are manipulating the stack at compile-time, modifying the compiled branch structures. Such tricks reduce readability and modifiability, increase complexity, are neither "standard" nor transportable to non-FIG systems, and are not necessary.

```
SCR # 12
  0 (  NEW,  A full screen editor                     WFR-79JUN16 )
  1 FORTH   DECIMAL
  2 : NEW      (  line #  ---     builds from this line, downward )
  3     16  0
  4   DO  CR  I  3  .R  SPACE
  5        I  OVER  -
  6        IF  [ DROP ] ( error )  QUERY  1  TEXT  PAD  1+  C@
  7           IF ( not at null )  I  EDITOR  R  FORTH  1+
  8              ELSE ( before or after )  8  EMIT  [ ROT 2 ]
  9         THEN     I  SCR  @  .LINE
 10              THEN
 11     LOOP  DROP  ;  CR  ."     NEW is loaded  "  ;S
 12 This editor builds a NEW screen.  Either list the screen or
 13 set SCR manually.  Then give:   'n  NEW'   where n is the first
 14 new line.  Previous lines are listed; an empty line will
 15 terminate building the new screen.
```





Let's try modifying the flowchart to make it structured. Repeating "print line" under the 2 top decision boxes makes this proper. A different kind of flowchart prevents this kind of error and is ideally suited to FORTH. It is called D-charts and was described in FORTH DIMENSIONS, Vol. 1, No. 3. Not only is a D-chart inherently structured, but also there is a one-to-one correspondence between the chart symbols and FORTH words. In the D-chart of NEW, the correspondence between symbols and words is as follows:

We will certainly want to use as much of the existing editor as we can to reduce our work. The line Replace and Insert commands are good candidates:

R	line# - Replace line with text from PAD.
I	line# - Insert the text from PAD at line line#, old line line# and subsequent lines are moved down. Line 15 is lost.

We can use FORTH as a Program Design Language (PDL) by:

1) starting with the top word (e.g., NEW or UNDER),

2) making up names for lower words (i.e., forward references),

3) and using the postfix order and FORTH control structures but not worrying about correct stack manipulation.

Later the result can be finished by defining all the words used, supplying necessary stack manipulation operators, and typing them in and debugging each in bottom-up order.

From the previous D-chart we could write the following pseudo-definition for NEW:

```
    : NEW                          16  0
DO
                               CR   .LINE#
                                 ENTER? IF
                 ENTER  NULL? IF
                    .LINE   ELSE
              (EDITOR's) R  THEN
                                       ELSE
                               .LINE  THEN
                                           LOOP
    ;
```

This incomplete definition does not take care of passing data on the stack or switching vocabularies. Look at the other command UNDER. The only change needed to the above code is to use the EDITOR's I instead of R. Because the two definitions are so similar, we will want to share some of the common parts.

To finish the definition of NEW, let's consider each undefined word.

.LINE#

needs to print the current line number right justified in 3 columns followed by a space. But should the line# be passed as a stack argument? The following definition sets it from the stack:

```
; .LINE#  (  line#  -  )  3  .R
SPACE  ;
```

The FORTH word I could be used before the reference to .LINE# in NEW's definition to supply the DO-LOOP index (which is the current line number). But what about using I inside .LINE#'s definition instead? Unfortunately it's not the same. In fig-FORTH DO keeps its indices on the return stack, so I doesn't return the index in another definition even though it was called from a DO-LOOP body. Another word which does that is called I' (pronounced I prime). Then .LINE# could be written:

```
: .LINE#  ( - )  I' 3 .R
   SPACE   ;
```

A high level definition for I' is:

```
: I'
  FORTH R> R> R ROT ROT >R >R  ;
```

(A CODE definition would be preferred.)

Considering the inefficiency of I' and readability, let's pass the line number on the stack.

The next choice is should we use a separate definition for .LINE# (as above) or copy the contents of its definition into NEW. Execution speed would be indistinguishable. Using the name .LINE#

might be more readable, but not much. The dictionary sizes are different for the two choices. (Sizes are in bytes.)

Passing I on the stack would make ENTER? look like:

```
: ENTER? (start-line# current-line#-)
     OVER =  ;
```

	.LINE# separate		included in NEW & UNDER
literal 3		4	2 x 4 = 8
.R SPACE		4	2 x 4 = 8
.LINE# head	5 + name size=	10	
;		2	
references	2 x 2 =	4	
		24	16

So for only 2 references to .LINE#, it doesn't pay to define it separately. (3 references would make it close: 24 to 26 bytes.)

ENTER?

This should be true:

1) when the current line # equals the starting line #

2) while new text is being entered

3) but not after a (CR) only has been entered.

We never want to use a VARIABLE for temporary storage if we can help it. The starting line number comes in from the stack, so (1) is simple

```
start-line#  I  =
```

(The argument must be preserved each iteration, so a DUP must be added; a DROP will have to follow LOOP to compensate.) Case (2) can be achieved by incrementing the start-line# while in enter-mode. This can be done with a 1+ after the Editor's R. Finally (3) falls out by not incrementing it after either .LINE in NEW's definition.

But more words are needed in NEW's definition to complete the enter-mode control. As with .LINE# before, the contents of ENTER? could be copied in NEW's definition instead of being defined separately. The size tradeoffs would favor that, but in this case readability would be greatly enhanced by keeping the name. This also eliminates the need to comment each part of that IF structure (as in the version on Screen 12).

ENTER

must wait for terminate input, then copy the entire line to PAD for later use by the editor.

QUERY reads a line of input, and TEXT can copy it to PAD:

TEXT c -
Copy text from the Terminal Input Buffer to PAD until the delimiter c is found.

So we could define ENTER with:

```
: ENTER  ( - )
  QUERY  1 TEXT  ;
```

NULL?

should be true only if a (CR) was ENTERed. fig-FORTH puts a null character (i.e., binary zero byte) in the Terminal Input Buffer (TIB) when a (CR) is entered. To tell if it is at the start of the buffer, we can use:

```
: NULL? ( - f )
  TIB @  C@  0= ;
```

Although keeping this definition separate would take up more space than using its contents inside NEW and UNDER, readability is improved, so we'll keep it.

Finally, .LINE

needs a screen number and line number. The line number can be supplied by the DO-LOOP index. So before each .LINE in NEW or UNDER add:

```
I   SCR @   .LINE
```

Incorporating all the above refinements into the previous pseudo-definition of NEW produces the following code:

The only remaining changes needed concerns vocabularies. To add these definitions to the EDITOR vocabulary, use the phrase

EDITOR DEFINITIONS

before the first definition, and the phrase

FORTH DEFINITIONS

after the last. But within NEW's definition we need to specify which I and R are intended. FORTH uses pairs of names to resolve such ambiguities. It's like last names in people's proper names:

JOHN DOE

JOHN DEERE

But in good postfix style, the vocabulary name must precede the word it applies to, and remains in effect until changed. Vocabulary names in fig-FORTH are IMMEDIATE, so they can be used inside definitions the same way as outside. Within NEW's definition, we need to insert FORTH before DO to make sure all the I's are DO-LOOP words and not editor words.

```
: ENTER?   ( start-line# current-line# - f )   OVER =   ;

: ENTER    ( - )   QUERY   1 TEXT   ;

: NULL?    ( - f )   TIB @   C@   0=   ;

: NEW   ( start-line# - )                    16 0 DO
                                     CR   I 3 .R   SPACE
                                           I ENTER? IF
                                     ENTER   NULL? IF
                                I   SCR @   .LINE   ELSE
                      I   ( EDITOR's )  R   1+   THEN
                                                  ELSE
                                  I   SCR @   .LINE   THEN
                                                      LOOP
           DROP   ;
```

Also we need to put EDITOR before the R (the editor's Replace command), and FORTH after R to make the remaining I's be DO-LOOP words.

Adding the vocabulary names makes the previous definitions testable. Trying them reveals that it all works except the line printed after the (CR) only was entered (i.e., leaving enter-mode) has one additional space before it. This skews that line from all the others. This is because fig-FORTH echos a space when the (CR) is entered. To fix this ugliness, back up the cursor 1 column before printing that line. For most terminals, a Back Space character will do the trick. (Not so on a memory-mapped terminal.) Defining the following will output a Back Space:

```
: .BS   ( - )   8 EMIT   ;
```

It should be inserted after the phrase NULL? IF in NEW's definition. Because this function is terminal-dependent, it definitely should be a separate definition.

The final working version follows:

```
SCR # 45
( EDITOR EXTENSIONS: NEW UNDER   KRH 9FEB81 )
EDITOR DEFINITIONS
: ENTER?   ( start-line# current-line# - f )   OVER =   ;
: ENTER   ( - )   QUERY   1 TEXT   ;
: NULL?   ( - f )   TIB @   C@  0=   ;
: .BS   ( - )   8 EMIT   ;

: NEW   ( start-line# - )   FORTH   16 0 DO   CR   I 3 .R SPACE
   I ENTER? IF   ENTER   NUL? IF   .BS   I  SCR @  .LINE   ELSE
      I   EDITOR  R  FORTH  1+   THEN   ELSE   I   SCR @  .LINE
   THEN   LOOP   DROP   ;
: UNDER ( start-line# - ) FORTH 1+   16 0 DO   CR   I 3 .R SPACE
   I ENTER? IF   ENTER   NULL? IF  .BS   I  SCR @  .LINE   ELSE
      I   EDITOR  I  FORTH  1+   THEN   ELSE   I   SCR @  .LINE
   THEN   LOOP   DROP   ;
FORTH DEFINITIONS
```

NEW PRODUCT

HOME GROWN APPLE II SYSTEM:

As an avid FORTH user, I would like to share my work with other Apple II users. Assembling the fig-FORTH model source code on CP/M and other systems with assembly language development tools is relatively straight forward, but for the primarily turn-key Apple a lot of additional, undocumented information is required. To equalize this situation I will supply my home grown Apple II system on disk to anyone for $30.00. No documentation, support, or instruction is provided save for technical notes on the disk supplementing the FIG installation manual. An assembler, screen editor, source code and associated compiler are included. The idea is to be able to upgrade and patch the system in various ways from listings (standards, anyone?). Not for beginners, not a commercial product, at your own risk. Contact George Lyons, 280 Henderson St.; Jersey City, NJ 07302

TO VIEW OR NOT TO VIEW (TO VIEW OR TO VIEW NOT?)

George William Shaw II

Sometime back, about one year ago, a fig-FORTH package was distributed to the members at the monthly FIG meeting. One of the programs in the package was a command called VIEW. This command would allow you to find the source text for a compiled definition and list it on the screen by simply typing VIEW, followed by the name of the command you wish to see the source text of.

I have been asked by Carl Street, the guest editor for this issue of Forth Dimensions, to write a commentary on this command which is to describe how the code originally submitted in the goodies package works and what other additions or changes I would make to the code.

So why have VIEW? VIEW adds convenience to writing and editing programs. The command allows you to get directly back to the source screen of a compiled definition, rather than trying to remember just what screen it was on. Most of us can remember approximately what screen or screens we have been working on, but if we have been working with more than a few screens, we would usually have to list a couple of screens to find the source to review or edit a given definition. VIEW eliminates this problem by allowing us to reference the source on the disk by the name of the compiled definition.

VIEW also takes very little system overhead. The entire compiled source for VIEW with all extensions mentioned in this article takes less than 170 bytes on my system. The compiling overhead is just as small. Only one or two bytes per definition and a negligible addition to compile time. Very inexpensive for the convenience and power it gives.

In order for VIEW to work, some of the resident defining words must be redefined. In pre-compiled fig-FORTH systems, the defining words CONSTANT, VARIABLE, VOCABULARY, : (colon), and <BUILDS must be redefined to contain a word called >DOC<. >DOC< will store in memory the disk screen number which contains the source of the definition being compiled. (On systems which can recompile themselves, >DOC< need not be placed in each one of the defining words. It need only be placed in the word CREATE, which is used by each of the defining words to enter a definition into the dictionary.)

With >DOC< in either CREATE or each of the defining words, the disk screen number which contains the source will be stored in memory to allow later referencing by the command VIEW. The command VIEW, then, has the task of finding the requested definition in the dictionary, fetching the screen number from memory, and listing the screen. This entire procedure is quite simple in FORTH and can be accomplished in a single line of source code (excluding the comment):

```
: VIEW      ( list source screen of definition )
       [COMPILE]  '  NFA  1  -  C@  LIST  [COMPILE]  EDITOR ;
```

The word [COMPILE] causes the word ' (tick) to be compiled into memory, rather than being executed at compile time (' is immediate). When VIEW is later executed ' will search the dictionary for the name which follows VIEW. NFA takes the address left on the stack by ' (the parameter field address) and changes it to the Name Field Address. 1 - then gives the address of the byte immediately preceding the name field. C@ extracts the screen number (which was stored at compile time) where the source of the definition is located. LIST prints the source of the definition. The second [COMPILE] allows EDITOR to be compiled (EDITOR is immediate) to select the editor vocabulary when VIEW is executed. This last step allows convenient entry into the editor for editing if desired.

```
    :  >DOC<   BLK @ B/SCR  /  C,  ;
```

>DOC< stores a one byte screen number in memory of the screen from which source text is currently being interpreted or compiled. BLK contains the block number as above. In fig-FORTH, the block number and the screen number may not be the same (there may be several blocks per screen), so a division is performed with B/SCR (blocks per screen) to obtain the screen number. If in your system B/SCR is one (1), you may eliminate the division by B/SCR and additionally speed the execution of >DOC<.

```
: CONSTANT     >DOC<  [COMPILE]  CONSTANT    ;
: VARIABLE     >DOC<  [COMPILE]  VARIABLE    ;
: VOCABULARY   >DOC<  [COMPILE]  VOCABULARY  ;
: :            >DOC<  [COMPILE]  :           ;
: <BUILDS      >DOC<  [COMPILE]  <BUILDS     ;
```

>DOC< is then placed immediately preceding each defining word to store into memory the screen number currently being interpreted. Since for most of us our fig-FORTH is pre-compiled (we can't recompile the basic FORTH system), each defining word is simply redefined to be preceded by >DOC<. The [COMPILE] in each of the words is actually only necessary in the redefinition of : (colon) because it is immediate and would attempt to execute at compile time rather than being compiled as desired. The other words are not immediate and would not have this problem.

Now, when any one of the defining words executes, >DOC< is executed, storing the screen number being compiled immediately preceding the name field of the definition. The area immediately preceding the name field was selected because this area can be addressed directly with existing FORTH words. The parameter field area of FORTH words is of variable length, so the area immediately following the end of the definition would not be as easily addressed.

When it is desired to VIEW a view-compiled word, the source screen number can easily be accessed and the definition listed. If a word which has not been compiled with the screen number preceding it is VIEWed, the screen determined by whatever byte immediately precedes the definition will be listed.

The current definition of VIEW works great except for a few minor idiosyncrasies. First, only a single byte is stored in memory for the source screen number. If you have screens above 255 and compile from them, the source cannot be viewed directly. A larger number is then needed. By simply changing the C, and C@ to , and @ in >DOC< and VIEW respectively, any screen currently accessible by the FORTH system could be VIEWed. Note that the address calculation must also be changed from 1 - to 2 - to account for the additional byte, as shown below:

```
: VIEW       ( list source screen of definition )
        [COMPILE]  '  NFA  2  -  @  LIST  [COMPILE]  EDITOR  ;
: >DOC<          BLK  @  B/SCR  /  ,    ;
```

Also, to the list of words being redefined I would add USER, CODE and CREATE. Redefining USER will allow the location of the definition of the user variable. Redefining CODE will allow the VIEWing of words defined in assembler. Redefining CREATE will cause all defining words later compiled to build VIEWable words.

```
: USER      >DOC<  USER    ;
: CODE      >DOC<  CODE    ;
: CREATE    >DOC<  CREATE  ;
```

It should be noted also that if you have changed the structure of your dictionary by placing links first (as I have) that the address calculation in VIEW will have to be changed as below:

```
: VIEW        ( list source screen of definition )
        [COMPILE]   ' NFA  4  -  @  LIST  [COMPILE]  EDITOR  ;
```

The additional 2 - (to 4 -) is necessary to skip the link which precedes (rather than follows) the name field in these systems.

And lastly, the current definition of VIEW will even try to list the source screen for definitions which have been created at the keyboard. The block number stored for these definitions is zero (0), which is not where the source is at all. If you don't mind having block zero (0) listed when you request to VIEW a definition which you created at the keyboard, then there is no problem. But, if this does bother you, you can put in the test below:

```
: VIEW       ( list source screen of definition )
        [COMPILE]  '  NFA  2  -  @
        -DUP IF   LIST [COMPILE] EDITOR   THEN   ;
```

In addition to the above, a test may be put in >DOC< to prevent the storing of the screen number when compiling from the keyboard:

```
: >DOC<       BLK @ -DUP IF R> R [illegible]     , THEN  ;
```

FIGURE 6

Note that if the test for block zero (0) is placed in >DOC<, then VIEW will try to list those definitions which would have had a screen number of zero (0) with the same result as attempting to VIEW a definition which was not defined with the redefined defining words.

George W. Shaw II
SHAW LABS, LTD.
P.O. Box 3471
Hayward, CA 94540

NEW PRODUCT

FORTH-79 FOR APPLE:

MicroMotion has announced the release of FORTH-79 for the Apple computer. MicroMotion FORTH-79 is a structured language that is claimed to conform to the new FORTH-79 International Standard. MicroMotion FORTH-79 comes with a screen editor and macro-assembler. Vocabularies are included for strings, double precision integers, LORES graphics and modem communication. Its operating system allows multiple disk drives and is 13 or 16 sector disk compatable. MicroMotion FORTH-79 runs on a 48K Apple II or Apple II Plus. Retail price is $89.95 including a professionally written tutorial and user's guide designed to make learning FORTH-79 easy for the beginner. MicroMotion; 12077 Wilshire Blvd., Suite 506; Los Angeles, CA 90025; (213) 821-4340

(Editor's note -- The manual is excellent. It notes the differences between fig-FORTH and FORTH-79 where pertinent)

SEARCH

John S. James

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. Incidentally, 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. Incidentally, 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!

Example of Use:

```
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
```

CORRECTION:

CROMEMCO DISKETTES described on page 145 of Vol. II/5 are supplied by:

Inner Access Corp.
PO Box 888
Belmont, CA 94002
(415) 591-8295

ARE YOU A — — — — — FIGGER?
YOU CAN BE!
RENEW TODAY!

GREATEST COMMON DIVISOR

Robert L. Smith

The problem of finding the greatest common divisor (GCD) of two integers was solved by Euclid more than 2200 years ago at the great library in Alexandria. The technique is known to this day as Euclid's Algorithm. The method is essentially an iteration of division of a prior divisor by a prior remainder to yield a new remainder. The quotients generated by this process are useful in other applications, such as rational fraction approximations, but are not required for finding the greatest common divisor.

For readers unfamiliar with the process, an example should clarify the method. Suppose we wish to find the GCD of 24960 and 25987. Divide one number into the other, and find the remainder or modulus:

```
25987 24960 MOD -> 1027
```

Divide the previous divisor 24960 by the remainder 1027 to yield:

```
24960 1027 MOD -> 312
```

Continue the process as follows:

```
1027 312 MOD -> 91

312 91 MOD -> 39

91 39 MOD -> 13

39 13 MOD -> 0
```

The last non-zero remainder is our desired answer, 13. This process must converge since the remainder is always less than the divisor. The process will terminate for finite numbers and integer division.

On Screen 20, we see a version of the greatest common divisor routine called G-C-D written in fig-FORTH. Line 1 begins a colon definition. In lines 2 and 3 the two arguments at the top of the stack are conditionally swapped to force the larger of the two arguments to be the first dividend. This step is used to avoid an unnecessary division in the succeeding part. However, lines 2 and 3 can be omitted entirely with no effect on the answer. The body of the calculation is in lines 4-7. At the start of the BEGIN-WHILE-REPEAT loop, the top element of the stack is the prior divisor and the second element is the prior remainder. In line 4 the new divisor (prior remainder) is saved, and the order of the top two elements reversed to prepare for the division. In line 5 the division with remainder is performed, and the remainder copied to the top of the stack for testing in line 6. For cases of non-zero remainders, the quotient is discarded but the remainder is kept in preparation for the next stage in the loop. The process terminates with a zero remainder. At line 8 the final quotient and remainder are dropped to yield the preceding remainder, which is the desired answer. Finally, the answer is printed out. The semicolon at the end of line 8 terminates the definition.

When Screen 20 is loaded, lines 9-11 are executed to print an invitation to the user to try the routine.

There are three areas in which this routine can be improved. The first is to remove lines 2 and 3 entirely, since the code does not usefully contribute to the final result. Furthermore, there is probably not even a speed advantage for machines with a hardware divide. Secondly, since the quotient is not used, the /MOD function can be replaced by the MOD function with a little reworking of the code. Finally, the printout function can be separated from the calculation function. It is usually advantageous in FORTH to write each definition so that it does as little as possible! The advantage of the separation in this case is that the calculation function can be applied repeatedly for finding the greatest common divisor of more than two arguments.

Our modified screen is shown below:

```
: GCD
     BEGIN
          SWAP OVER MOD ?DUP 0=
     UNTIL
     ;

: G-C-D
     GCD CR ." The G-C-D is " .
     ;

CR ." Input two numbers, then"
CR ." execute 'G-C-D'.  The"
CR ." greatest common divisor"
CR ." of these numbers will be"
CR ." displayed."
CR
```

The sequence in GCD is quite easy to follow now. The two arguments on the stack are swapped, and the 2nd element is copied over the first, in preparation for the division implied by the MOD function. The word ?DUP is the 79-Standard version of the fig-FORTH word -DUP. The function of ?DUP is to duplicate the top element of the stack (the remainder from the division in this case), but only if the remainder is non-zero. The function 0= reverses the logical value of the top stack element, so that the test in UNTIL will cause a branch back to the BEGIN part when the MOD function results in a non-zero value. When the remainder is zero, the zero value is not duplicated. Instead, the 0= function converts it to a 1, which in turn is dropped by the action of UNTIL. Furthermore, control is then passed from the BEGIN-END loop, and the function terminates, leaving only the previous non-zero remainder.

Note that the number of FORTH words in the basic definition has been effectively cut in half, compared to the original version in Screen 20.

The author gratefully acknowledges discussions with LaFarr Stuart in the preparation of this article.

```
SCR # 20
 ( Greatest common divisor, a demo         WFR-79DEC09 )
 : G-C-D
   OVER OVER <
   IF  SWAP  THEN  ( use larger as quotient )
   BEGIN  SWAP  OVER     ( save divisor third )
          /MOD  OVER     ( test remainder zero )
     WHILE  ( not zero )  DROP  ( this dividend )
    REPEAT
          DROP  DROP  CR ." The G-C-D is " . ;
 CR ." Input two numbers, then execute 'G-C-D'.  The greatest"
   ." common divisor of these numbers will be displayed."
 CR


 ;S

```

PROGRAMMING HINTS

PROGRAMMING APPLICATIONS, DEMONSTRATIONS & EXPLANATIONS

```
 0 ( ::     by guest writer Henry Laxen              Feb 81   )
 1   HERE >R      ( Save current DP on return stack           )
 2   [ ' QUIT CFA @ ] LITERAL ,  ( Get runtime for :          )
 3   !CSP ]       ( Security, start compiling                 )
 4   BEGIN
 5     INTERPRET    ( Compile what is typed                   )
 6     STATE @ WHILE  ( Until state changes                   )
 7     CR QUERY     ( Get another line from TTY               )
 8   REPEAT
 9   SMUDGE       ( Undo what ; did                           )
10   R> EXECUTE   ( Now do what user wanted                   )
11   R> DP ! ;    ( And restore dictionary                    )
12
13
14
15
```

:: is an excellent example of the flexibility of FORTH. Certain constructs in FORTH cannot be typed in from the terminal unless the user is in compilation mode. These constructs include: DO, LOOP, IF, ELSE, THEN, and all of the conditional compiling words. However, :: allows you to do this if you so desire. The idea is simple enough; you create an "orphan" word in the dictionary, execute it, and then forget it. (An orphan is a definition without a name header).

Let's step through the above definition line by line and see what is happening at each point:

1 HERE >R

HERE is the location of the next available dictionary entry. This location is saved on the return stack so it can be restored later.

2 [' QUIT CFA @] LITERAL ,

[changes from compilation mode to interpretation mode. QUIT has been previously defined as a high level : definition, and hence we use ' QUIT to get the address of its PFA. The CFA then converts this PFA to the CFA for QUIT. Since QUIT is a : definition, this CFA points to the runtime for : , which controls the nesting level in FORTH. The @ gets this address and places it on the parameter stack. Now the] places us back into compilation mode. The value we have thus computed, namely the runtime address of : , is then compiled as a literal in the definition for :: . When :: is executed, this literal is compiled inline by the , that follows. This has set up what follows as a : definition, so that it will execute properly when the time comes.

3 !CSP]

The !CSP is used for compile time error checking. The check is made when the user types ; to end his definition. The] puts the user into the compile state. What is typed from now on will be compiled instead of interpreted.

4 BEGIN

This denotes the beginning of some kind of looping structure.

5 INTERPRET

This is the main word in FORTH. It either executes or compiles the words it encounters depending on the current state. In our case it is compiling the words it encounters.

6 STATE @ WHILE

STATE is the variable which determines whether one is interpreting or compiling. When it is non-zero one is compiling, hence the loop is repeated as long as the user is still compiling. When you type ; STATE is set to zero, and this loop is exited.

7 CR QUERY

This simply gets another line from the terminal so that INTERPRET can compile it.

8 REPEAT

This is the end of the BEGIN loop.

9 SMUDGE

SMUDGE is used to undo the SMUDGE present inside of ; . It has no other purpose in this context.

10 R EXECUTE

This executes the word we have been building until now. If all goes well it will return.

11 R> DP !

And now we restore the dictionary to its previous state.

Note that there are still things you should not do with this implementation of :: , namely if what you are executing alters the dictionary, say by compiling additional words, the system will crash. An interesting exercise for the reader would be to redefine :: so that this is not the case.

This article contributed by Henry Laxen; 1259 Cornell; Berkeley, CA 94706

NEW PRODUCT

GO-FORTH FOR THE APPLE II:

The CAI FORTH Instruction System by Don Colburn is now available for the Apple. The GO-FORTH CAI System takes the novice FORTH programmer through the pitfalls of learning FORTH and lets him fly. Requires 48K Apple II plus Apple disc. Price is $45.00 per system (1-3 units), $30.00 per system (more than 4 units). International Computers; 110 McGregor Avenue; Mt. Arlington, NJ 07856; (201) 663-1580 (evenings).

NEW PRODUCTS

CROSS-COMPILER PROGRAM:

Nautilus Systems now offers a cross-compiler program for FORTH users. Machine readable versions are now available for the following hardware systems: LSI-11, CP/M, TRS-80, Apple, H-89, and Northstar. Each version includes: An executable version of figFORTH model 1.0; Cross-compilable source; Utilities; and Documentation. The cross-compiler is written entirely in high level figFORTH. Progam features include: Automatic forward referencing to any word or label; Headerless code production capability; ROMable code production capability; Load map; Comprehensive list of undefined symbols. Price is $150.00 including shipping. (California residents please add sales tax). Nautilus Systems; P.O. Box 1098; Santa Cruz, CA 95061; (408) 475-7461

TIMIN ENGINEERING FORTH:

Timin Engineering is now offering a version of figFORTH for 8080/8085/Z-80 or CDOS systems with at least 24K of memory. The Timin system features a FORTH style editor with 20 commands, a virtual memory subsystem for disk I/O, a Z-80/8080 assembler, and an interleaved disk format to minimize disk access time. Documentation includes a manual that may be purchased separately for $20 (credited towards purchase of disk). Price $95.00 on IBM compatible 8 inch single density disk (other disk formats $110) -- California residents please add 6% sales tax. -- and includes shipping by mail in U.S. Mitchell E. Timin Engineering Company; 9575 Genesee Avenue, Suite E-2; San Diego, CA 92121; (714) 455-9008

DEVELOPMENT OF A DUMP UTILITY

```
(DEVELOPMENT OF A DUMP UTILITY By John Bumgarner  March 81  )
OK
OK
 1  (DUMP MEMORY BYTES. ADDRESS COUNT. . .              )  OK
OK
 2  : DUMP  0 DO  DUP I + C@ 3 .R LOOP DROP ;  OK
OK
 3  HEX  OK
 4  1 2 3  HERE  10  DUMP  CR . . .  4 44 55 4D 50 20 20 20 20 20 20 20 20 20 20 20
 5  3 2 1  OK
OK
 6  ( Test for non-printing ASCII Character.  CH .. . T/F)  OK
OK
 7  : ?NON-PRINTING  DUP 20 <  SWAP 7E >  OR ;  OK
 8  : Q  ?NON-PRINTING . ;  OK
 9  -10 Q 0 Q 10 Q  1F Q  20 Q  40 Q 1 1 1 1 0 0  OK
10  7E Q 7F Q 100 Q  7FFF Q 8000 Q  0 1 1 1 1  OK
11  FORGET Q  OK
OK
OK
12  ( Type any memory bytes using  .  for non-printing characters)  OK
13  ( Address Count ...)  OK
OK
14  : &TYPE  0 DO  DUP I + C@
       DUP  ?NON-PRINTING IF DROP 2E ( .) THEN EMIT LOOP DROP ; OK
OK
15  1 2 3 HERE 10 &TYPE CR . . . .&TYPE
16  3 2 1  OK
OK
OK
17  (Print address , DUMP &TYPE 16 bytes.  Address ... )  OK
OK
18  : A-LINE  CR DUP 0 6 D.R SPACE 10 OVER OVER DUMP 2 SPACES &TYPE ;  OK
OK
19  HERE A-LINE
     2419  6 41 2D 4C 49 4E 45 20 20 20 20 20 20 20 20 20     .A-LINE  OK
OK
OK
20  2 3 4 HERE A-LINE CR . . .
     2419  6 41 2D 4C 49 4E 45 20 20 20 20 20 20 20 20 20 .A-LINE
4 3 2  OK
OK
OK
21  (Can't think of a better name.  Address count ... )  OK
OK
22  :  DUMP  0 DO DUP I + A-LINE 10 +LOOP DROP ; DUMP isn't unique OK
OK
23  HERE 40 DUMP
      243A  4 44 55 4D 50 20 20 20 20 20 20 20 20 20 20 20 .DUMP
      244A 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
      245A 20 20 32 84 44 55 4D D0 5F 5F 5F 5F 5F 5F 5F F5   2.DUM._          _
      246A 5F 5F 5F 5F 5F 5F 5F 5F 5F 5F 5F 5F 5F 5F 5F 5F _                 _  OK
OK
OK
```

Program development in FORTH can --and should--be done in a "top down" manner as this type of design produces error free programs in a minimum of time. However, the road to a solution is not always direct; and experienced FORTH programmers often play with a programming problem on the terminal to get ideas. These idea sessions usually result in the data necessary for "top down" program design. The development steps involved in producing a really useful tool from a very simple DUMP word will illustrate just such a design session.

I begin by setting down in the comment on line 1 the functions and parameters for the fundamental DUMP word to be defined -- I simply want it to dump values from memory. Given the starting address and number of bytes to dump as parameters on the stack, the word defined in line 2 is short and simple. The count value is on top of the stack which suggests using a DO ... LOOP to do the work. Inside the loop we put the code to generate an address, fetch and print a byte. The loop parameters are the count value on the stack and the zero put inside the definition just in front of the DO.

FORTH DO loops increment the loop index and check it against the limit at the end of the loop. If the newly incremented index is less than the limit, the loop body is executed again. If the index is greater than the limit, the loop is terminated and execution continues with the next word. This method of loop control results in the loop index (fetched for use by the word I) going from 0 to count 1- (that is Count-1 for non-FORTH persons) by 1's for this FORTH DUMP word. The loop index then is just what we need to add to the starting address to obtain successive byte addresses for dumping.

The inside of the loop first duplicates the address (the only thing remaining on the stack after DO removed its two arguments) to save a copy and then adds the loop index to it. Using this new address we fetch a byte with C@ and print it, right justified in a 3 column field using 3 .R . In the same manner we keep looping (incrementing the index by 1 each time) adding, fetching and printing. When the loop index equals the count, the loop exits and drops the extra copy of the address to clean up the parameter stack--A very important step!

FORTH is highly interactive and nothing is more natural than to test the new DUMP word immediately. In line 3 I switch to hexadecimal I/O to enable easy interpretation of the dump results in line 4. Before I actually execute DUMP in line 4 I put a few numbers on the stack so that I can check to see that the stack is not altered by DUMP . The word HERE provides a starting address for DUMP and then I select 10 (in decimal, 10 hex is 16) bytes to dump. The CR . . . after DUMP prints (on a new line) and removes the top three numbers of the stack. Lines 4 and 5 show the execution of the word and also show my check numbers are where they should be. This simple check shows that DUMP is very likely to be free of errors and we may proceed with confidence.

I happen to like dumps that dump both byte values and ASCII characters and so my next task was to make a word that could dump in ASCII what DUMP output as numbers. I have also learned from frustating experience that printers and terminals should not be given byte values that are non-printing characters lest they do some strange things in response. By thinking ahead a little bit I can see a definite need for a word to alter the memory byte values to an acceptable range for the output device. The result of this advanced thinking is shown on lines 6 and 7. Given a character value on the stack ?NON-PRINTING will return a True (plus one) value if it is a non-printing character and an False (zero) otherwise. The test is made for values less than hex 20 (a space) and greater than hex 7E (a tilde).

At this point I am still in hex for numeric I/O (see line 3) and so I use 20 and 7E directly in the definition without having to look up their equivalent decimal values. The OR at the end of ?NON-PRINTING combines the results of the two limit checks and produces a single true/false value for the output if either limit is exceeded.

The verification of a limit checking word like ?NON-PRINTING requires careful testing with data that exercises the word over its entire range and especially at critical places such as the limits. With ?NON-PRINTING this means testing 10 or more times with carefully chosen input values. The name of ?NON-PRINTING is nice and descriptive but is too long for a poor typist like me to type often so on line 8 I have defined the word Q to execute ?NON-PRINTING and print the result. On lines 9 and 10 I try my new Q word out with some numbers designed to make it fail if it will. Numbers such as small negatives, zero, small positives, on either side of the lower limit, mid range, either side of the upper limit, larger positives and at the magic place where the sign bit changes value. The results show that ?NON-PRINTING is not fooled by these tests and so on line 11 I discard my test word Q .

On lines 12 and 13 I put down my ideas for the word which will type the ASCII character equivalents for my memory byte values and begin the definition on line 14. This definition is just like the one for DUMP except for what happens to the byte value after it is fetched. In place of the 3 .R of DUMP we check for non-printing characters with ?NON-PRINTING and if they are non-printing we DROP the value and replace it with the hex code for a dot, 2E. Then we EMIT the byte to see the ASCII character for that byte's value.

Some programmers will want to mask the byte value with hex 7F to zero the high bit before testing for non-printing characters. One reason to do this is so fig-FORTH names will show up better in dumps since the "traverse" bit on the last kept name character would otherwise turn that character into a non-printing value and it would appear as a dot. My reason for not doing it is that getting rid of the high bit turns too much data into ASCII and clutters up that part of the dump. The testing of &TYPE is done exactly like the testing for DUMP , the .&TYPE after the CR . . . is the output of &TYPE . The dot before the ampersand is the count byte of the string getting turned into a dot by ?NON-PRINTING .

Incidentally, the string found at HERE when a given word is executing in the interpretive mode (outer or name interpreter in this case) is the name of the word. Look up the ASCII characters for the byte values dumped in line 4 and you will find they spell DUMP . This bit of magic is performed by the word WORD in figFORTH.

With the two words DUMP and &TYPE we are in a good position to write a word that will dump one line of information on the terminal. This word can then be executed in a loop, once for each line dumped. I call this word A-LINE and define it in line 18 after noting in line 17 that it should print the starting address of the dumped line. A normal line width on a CRT type terminal is 80 columns and so there is room for 16 bytes dumped (3 16 * columns), typed (16 more columns) and an address and spaces for separators (9 columns). The room used adds up to 73 columns but this could be reduced a few columns. If the address were always printed as an unsigned hex number it would only take up 4 spaces. Then by allowing only one space between the three fields on a line the line width would be 70 columns (4 1+ 48 + 1+ 16 +). If you have a smaller output line then you must give up the ASCII characters or the spaces between the byte values or dump less bytes per line.

The actual work done by A-LINE is to first do a CR to get a new line to use and then output the address as an unsigned number, right justified in a 6 column field. The address output is done by the phrase:

```
DUP  ( Save a copy of the address )
  0  ( Put a zero on top of )
     ( the stack to make a positive )
     ( 32 bit number )
  6  ( 6 column field )
D.R  ( Output a 32 bit number )
     ( in a field )
```

Next we output a SPACE to help separate the address from the byte values; put 16 (16 is hex 10) on the stack and copy both the saved address and our count of 16 by using OVER OVER (if you have it use 2DUP) . The stack is now set up with two sets of addresses and counts ready for our two words DUMP and &TYPE . I put 2 spaces in to separate the byte values from the ASCII characters.

On lines 19 and 20 of the print-out I test A-LINE for stack problems and functionality and it seems that everything is working correctly.

At last I am now ready to write the actual dump word. The comment on line 21 starts me off with the desired parameters and a note showing my limited personal vocabulary. My inability to think of another name is not going to be a problem, however, since FORTH allows you to redefine names and use them over. All that happens is that a warning message (isn't unique) is displayed to alert you to the redefinition. The warning message appears here after the end of the definition on line 22. When the new definition with the re-used name successfully compiles, the earlier definition of the same name becomes unavailable for future use--either by compiling into new words or interpretively executed from the terminal. Since I do not want my old definition of DUMP on line 2 anymore, I consider this to be an advantage. I have an improved DUMP and I do not have to think of another name for it! What is more, everything will work properly as before because the use of the old DUMP in the definition of A-LINE does not get changed--what is compiled stays compiled as it was. All that happens as far as a user of DUMP is concerned is that if DUMP is now asked for, the new one will be found first automatically and the search will stop there -- the system never suspects that another, different version of DUMP is hiding down the dictionary a ways.

The actual definition of the new DUMP is very similar to the old DUMP except that we are substituting A-LINE for the C@ 3 .R and since we are dumping 16 bytes on a line we use +LOOP to terminate the DO and give it a 16 (10 hex) to add onto the loop index each time. Now the address computation done by the DUP I + phrase will start at the specified point and go up by 16 bytes each time through the loop.

The useful tool that I set out to develop at the start of this session is now complete and only needs to be tested. Line 23 does this final test, but because of my earlier successful test of A-LINE and also because the new DUMP is so similar to the old DUMP I did not bother to try putting some numbers on the stack to see if it adds or removes any values. I now have high confidence that DUMP will work correctly and it does.

This article contributed by John Bumgarner; FORWARD TECHNOLOGY; 1440 Koll Circle, Suite 105; San Jose, CA 95112

NEW PRODUCT

MICROPOLIS FORTH:

Acropolis now offers FORTH for Micropolis. Acropolis FORTH (A-FORTH) runs under Micropolis MDOS on 8080/8085 and Z-80 systems running at 2 or 4 MHz with 32K memory and at least one MOD I or MOD II disk. A-FORTH has 2 program/data file editors -- a line editor for standard serial terminals and a screen editor for memory-mapped terminals. A-FORTH has an 8080/8085 macro-assembler that allows use of any mixture of A-FORTH and assembly code desired in a single definition. A-FORTH has all the features of figFORTH plus: Double precision math & stack operations (32 bit); Double precision variables & constants (32 bit); Multi dimensional arrays up to the limits of available memory; Virtual arrays up to the limit of disk storage on all disks; Case statements; Printer support using MDOS ASSIGN statements; Forgetting across vocabulary boundaries; Enhanced disk procedures that reduce response time, compiling time, & number of disk accesses; Physical disk support for disk diagnostics and disk copy and direct access to MDOS file directory. Acropolis A-FORTH has an 89 page users manual. Acropolis provides A-FORTH updates & patches at no charge for 1 year after purchase. Price $150.00 including shipping (California residents add 6% sales tax). Acropolis division Shaw Labs, Ltd.; 17453 Via Valencia; P.O. Box 3471; Hayward, CA 94540; (415) 276-6050

NEW PRODUCT

ALPHA MICRO REENTRANT FORTH:

Sierra Computer Company is now offering version B of their AM-FORTH for Alpha Micro system AM-100 computers. Version B is said to be reentrant, allowing the basic FORTH dictionary to be loaded as part of the AMOS system and shared by any number of users in the multi-user Alpha Micro system. Other new features include: An assembler; Screen oriented editor; Support of special AMOS CRT handling features; Floating point math operations; Utilities for string handling and building data structures and access to system TIME and DATE functions; More versatile I/O to AMOS sequential and RANDOM files; and use of lower case characters. All features of version A are included in version B. AM-FORTH version B is available on AMS or STD disk that contains complete source code; executable object code; FORTH utilities for the editor, assembler and data structures and some sample FORTH programs. Complete documentaton describing AM-FORTH implementation, installation procedures, operating instructions and glossary. Price is $150.00 ($120.00 to licensed version A purchasers at $40.00). Contact George Young; Sierra Computer Company; 617 Mark NE; Albuquerque, NM 87123

LETTERS

Dear FIG,

I have been using the May '79 release of 6800 fig-FORTH since it was issued. A question that isn't apparent on the order form is, has a further release been made either on the assembly source listing or installation manual? This may be a saving for us by preventing a duplication of our software.

N.H. Champion
Prescott, AZ

Two small changes have been made to the FIG model. In screen # 23, U* has been corrected for a carry bug. Screens 93 and 94 have been converted from assembly to high level. The assembly listings have not changed in the last year. Here are the revision/publication dates for each FIG publication:

Publication	Release	Date
Installation Manual	1.0	11/80
Listing		
8080	1.1	9/79
6800	1.0	5/79
6809	1.0	6/80
6502		
9900	1.0	3/81
8086/88	1.0	3/81
PDP-11	1.3	1/80
PACE	1.0	5/79
ALPHA MICRO	1	9/80

Hope this clarifies your question. -- ed.

Dear FIG,

I would like to see programming examples as part of every meeting agenda. Perhaps a theme could be established for each meeting.

Also publishing programming examples would be helpful. I, for one, find examples the best method of learning and attempting to reach the point where I start building FORTH programs efficiently.

I also recognize that one person can't do it all, and that a successful users group depends upon contributions from everyone. I am not sure how I can help at this time, but I am willing to do my share.

J. Arthur Graham
Orinda, CA

Glad to hear it! See this month's edition for programming examples and this month's editor's column regarding helping out. -- ed.

Dear FIG,

I am interested in corresponding with others interested in FORTH on larger machines. I can be reached at the address below.

Stewart Rubenstein
HARVARD UNIVERSITY CHEMICAL LABS
12 Oxford St., Box 100
Cambridge, MA 02138

Dear FIG,

Would it be possible to include some tutorial articles on the inner workings of FORTH in FORTH DIMENSIONS?

Being new to this language, [I find] the functions of the interpreters and the compiler somewhat mysterious.

LETTERS

Given the extensibility of FORTH, a better understanding of the guts of the language is an advantage. I haven't found a publication that completely describes these functions.

R. Stockhausen
Milwaukee, WI

See Dr. C.H. Ting's SYSTEMS GUIDE TO fig-FORTH, available from FIG -- you can use the order blank at the back of this issue. -- ed.

Dear FIG,

Our membership is growing and I have delivered fig-FORTHs to several of the Black African countries.

Rhodes University has adopted 6809 fig-FORTH for its curriculum this year. UNISA will follow, God willing, next year in its microprocessor course, and several other universities are using various versions of fig-FORTH for research purposes.

I've written several articles, both local and abroad on FORTH and I'll send you copies of these. Please give us some support and coverage. I'll write you at least once a month.

Ed Murray
FORTHWITH COMPUTERS/FIGSA
P.O. Box 29452
Sunnyside, Pretoria, 0132
South Africa

Always happy to hear from our international contingent! Your meeting announcements are in our announcement section. -- ed.

DEA- FIG,

I AM A FOR-- PRO------- CUR------ EMP----- BY FOR-- INC. I HAV- NOT WOR--- ON ANY FIG TYP- SYS---- AND I AM EXC---- BY THE VAR----- LEN--- NAM- IDE-. PLE--- SEN- ME THE FIG FOR-- MOD-- SO THA- I MAY TRY IT OUT HER- AT FOR-- INC.

FRE- THO----

Your request is answered. Next edition you will be able to communicate with four+ letter words! -- ed.

Dear FIG,

I am a long time FIG member and am seriously devoted to FORTH as a programming language and system. Like many others, I have FORTH running now and after all the talk about how great it is, I find few (hardly any) complete examples of its use in solving real, practical problems.

The point of all this is a suggestion that FIG publish more articles and papers on practical applications -- programs which can be easily put into everyday use by any programmer. One can go to the magazine counter at any computer store and find many examples of practical programs in BASIC. FORTH should be even more appropriate for such applications.

I believe that the organization, and each of us as members, can contribute to this end. I propose FIG strongly solicit contributions of articles dealing with practical programming projects developed in FORTH.

George O. Young III
Albuquerque, NM

You took the words right out of our editorial mouths. We hear you and are looking forward to receiving contributions. -- ed.

Dear FIG,

I have developed a generalized data structure for vocabularies which removes many of the limitations now found in both FIG and other FORTH models.

My new structure has most of the advantages of the present FIG model, plus it allows multiple threads per vocabulary with different numbers of threads in each, if desired. With this multiple thread concept vocabularies are physically linked with a single pointer and are both sealed and linked simultaneously.

I have also developed a "vocabulary stack" to allow context specification in line with the FORTH '79 standard. I intend to make my findings available at the next FORML conference.

If anyone would like to contribute suggestions or developments along these lines (especially the vocabulary stack) for release in the public domain please write me at the address below:

EXIT (in line with the 79-standard)

George W. Shaw II
SHAW LABS, LTD.
P.O. Box 3471
Hayward, CA 94540

Dear FIG,

Help! A while back I got my copy of figFORTH-8080 version. I'm bogged down at the "MATCH" primitive of the "EDITOR" function. I'm working alone at it as home computers are rare up here and FORTH is a "Very Foreign Language". All I need to know is what in tarnation one uses to interpret screens 93 & 94 to 8080 (or Z80) code?

I have the rest of the FIG model working, although I've had moments with it ranging from tears to apoplexy. I've discovered some of the no-no's the hard way, also known as "How to reconfigure your disk -- unexpectedly." or "Where did the CP/M go?".

I haven't had so much fun since I built this "United Nations computer".

Regards,
Glenn Farnsworth
Weed, CA

Editor's note -- The editor was included with the model as an extra "goodie". A little foresight would have told us the 6502 assembly source would prove to be an irritant. The high level equivalent is given below. A full screen search with a code MATCH takes about 150 msec, while the high level form requires over a second. Try the high level version and then recode for your processor. This addition to the model was made in September 1980 thanks to Peter Midnight who provided an earlier definition.

Keep smiling! -- ed.

```
SCR # 148
( double number support                          WFR-80APR24 )
( operates on 32 bit double numbers or two 16 bit integers )
FORTH DEFINITIONS

: 2DROP    DROP  DROP  ;              ( drop double number )

: 2DUP   OVER  OVER  ;         ( duplicate a double number

: 2SWAP  ROT  >R  ROT  R>  ;
                  ( bring a second double to top of stack )
EDITOR DEFINITIONS  -->
```

```
SCR # 149
(  String MATCH for editor                    PM-WFR-80APR25 )
: -TEXT                   ( address-3, count-2, address-1 --- )
SWAP  -DUP IF  ( leave boolean matched=non-zero, nope=zero )
              OVER  +  SWAP   ( neither address may be zero! )
        DO  DUP  C@  FORTH  I  C@  -
            IF  0=  LEAVE  ELSE  1+  THEN  LOOP
        ELSE  DROP  0=  THEN  ;
: MATCH (cursor address-4, bytes left-3, string address-2,
          (string count-1, --- boolean-2, cursor movement-1 )
 >R  >R  2DUP  R>  R>  2SWAP  OVER  +  SWAP
(caddr-6, bleft-5, $addr-4, $len-3, caddr+bleft-2, caddr-1)
 DO  2DUP  FORTH  I  -TEXT
  IF  >R  2DROP  R>  -  I  SWAP  -  0  SWAP  0  0  LEAVE
       ( caddr bleft $addr  $len  or else 0 offset 0 0      )
   THEN  LOOP  2DROP  ( caddr-2, bleft-1, or 0-2, offset-1)
  SWAP  0=  SWAP  ;
```

ANNOUNCEMENTS

PREVIEWS OF COMING ATTRACTIONS (IN FORTH DIMENSIONS):

Issue	Editorial Content
May/Jun	Applications, utilities & useable programs
Jul/Aug	Games & game type applications
Sep/Oct	University of Rochester & Utrecht conferences
Nov/Dec	Graphics & music

If you would like to be a contributing author to any of the above please write to: Editor; FORTH DIMENSIONS; P.O. Box 1105; San Carlos, CA 94070. You will be sent a writer's kit that will make your job easier. Please note deadlines for each issue are several months in advance of publication dates so allow plenty of time to produce your article.

FIG GOES TO COMPUTER FAIRE: FIG will have booth number 1137C at the West Coast Computer Faire being held April 3 to 5 at Brooks Hall in San Francisco.

FREE BUG FIXES: The 8080 Renovation Project wants bug reports so they can get to work on fixing them. If you have found an 8080 Bug send it to 8080 Renovation Project; c/o FORTH Interest Group; P.O. BOX 1105; San Carlos, CA 94070

DR. DOBBS NEEDS YOUR HELP: The editor of Dr. Dobb's Journal of Computer Calisthenics & Orthodontia is very interested in articles on FORTH. If he can get enough, he will devote an entire issue to FORTH. Interested authors should contact Marlin Ouverson, Editor; PEOPLE'S COMPUTER COMPANY; PO Box E; Menlo Park, CA 94025

CALL FOR PAPERS

FIG STANDARDS TEAM: The FORTH Standards Team announces the Spring Conference hosted by the University of Rochester on May 13th through May 15th, 1981. Larry Forsley is the session organizer. This conference will have three components: Formal papers, Sub-team working groups, and Poster sessions.

Formal papers must be received by May 1st. Later material and informal presentations will be assigned to the "Poster session"; at which the authors will conduct clustered workshops, with attendees moving among the presentations. The Sub-teams will prepare short reports after topic oriented working sessions.

Working sessions are scheduled from the morning of May 13th through lunch on May 15th. A reception will be held on the evening of May 12th for early arrivals. Accomodations are $12.00 single occupancy and $9.00 each, double occupancy. A combination of campus and off-campus meals are planned.

Papers are specifically requested on:

1. Implemtation aspects of FORTH-79
2. Refinements of vocabulary structure, extensible control structures, definition of input and output streams.
3. File sytem extension
4. Floating point extensions

The contact for submittal of papers and room reservations is Larry Forsley; Laboratory for Laser Energetics; University of Rochester, NY 14623. Send room requests without delay; a confirmation with exact cost will be returned with the conference schedule and travel suggestions.

MEETING/EVENT ANNOUNCEMENT FORMAT

In order to have uniformity and insure complete information in all meeting and special event announcements, FORTH DIMENSIONS requests that you use the following format:

1. WHO is holding the event (organization, club, etc.)
2. WHAT is being held (describe activity, speakers' names, etc.)
3. WHEN is it being held (days, times, etc.; please indicate if it is a repetitive event -- monthly meeting etc.)
4. WHERE is it being held (be as complete as possible -- room number, etc.)
5. WHY is it being held (purpose, objectives, etc.)
6. REMARKS & SPECIAL NOTES (is there a fee, are meals/ refreshments being provided, dress, tools, special requirements, pre-requisites, etc.)
7. PERSON TO CONTACT
8. PHONE NUMBER/ADDRESS (include area codes, times to call & give work & home numbers in case we need clarification)

ATTENTION 6502 USERS:

The following seem to me to be errors in the 6502 Assembly Source Listing (May 1980). I think I can correct these errors easily enough, but I worry if maybe they have generated more subtle errors that I have not found. I have no experience with FORTH at all, so I'm not sure what should be happening, and I have no one I know with any experience to call upon.

Page 0061 UPDATE Missing SEMIS at end? (There <u>is</u> one in the installation manual)

0064 Line 3075 Shouldn't this be a backward branch with F6 FF as displacement?

0067 Lines 3204 - 3205 Two STX XSAVE's. Is one superfluous or is it replacing something else that really should be there?

0069 Lines 3280 - 3284 Two SEMIS. Again, is something being destroyed by the extra one?

C.A. McCarthy
Department of Mathematics
Vincent Hall
UNIVERSITY OF MINNESOTA
Minneapolis, MN 55455

RENEW TODAY!

MEETINGS

FORTH INTEREST GROUP U.K.: Chairman: Dick de Grandis-Harrison; Secretary/Treasurer: Harry Dobson; Newsletter Editor: Gil Filbey; Committee Members: Bill Powell, Bill Stoddart. Meetings are held at 7 p.m. on the 1st Thursday in every even month at:

The Polytechnic of the Southbank
Room 408
Borough Road
LONDON

Mailing Address:

FORTH INTEREST GROUP U.K.
c/o 38, Worsley Road
Frimley, Camberley,
Surrey, GU16 5AU
ENGLAND

PORTLAND FORTH USERS GROUP: Held its first meeting in January. Demos were given on an Apple II. Also shown were a Hires graphic package written in FORTH; A "de-FORTHer" program that takes FORTH words down to their component parts; and a 64 bit quad precision math package. FORTH concepts such as the word DEPTH and .S (a non-destructive stack print out) were also discussed. Meetings are held monthly at THE COMPUTER & THINGS STORE; 3460 S.W. 185th, Suite D; Aloha OR 97006

TULSA COMPUTER SOCIETY: A FORTH Interest Group has been formed in Tulsa, OK under the auspices of the Tulsa Computer Society. The group has 6502 figFORTH running on several Apple II's and 8080 figFORTH running on a Compucolor and a MITS Altair using CP/M and Micropolis Drives. For meeting information contact Art Gorski; c/o The Tulsa Computer Society; P.O. Box 1133; Tulsa, OK 74103 or call (918) 743-0113; (918) 743-4081

SOCIETE D'INFORMATIQUE AMATEUR DU QUEBEC: Has a FORTH group (French!) that meets every other week. Anyone from the Quebec area who would like meeting information is invited to contact Gilles Paillard; 1310 Des Pins Est; Ancienne-Lorette; Quebec, Canada G2E 1G2 or call (418) 871-1960

FIGSA: South Africa has a very active FORTH Interest Group meeting monthly and currently is offering FORTH mini-courses to ground users in the fundamentals. Interested persons in the Johannesburg and Pretoria locales can get more information regarding meetings and courses by contacting Ed Murray; FORTHWITH COMPUTERS; PO Box 27175; Sunnyside Pretoria 0132, South Africa

SOUTHERN CALIFORNIA fig: Attendees numbered approximately thirty-five and most had up and running FORTH systems. Three books were reported: Threaded Interpretive Language by Loeliger, which steps the reader through Z-80 source code of fig-FORTH for the TRS-80; MINT, Machine-Independent Organic Software Tools, by Godfrey, et al.; and FORTH SYSTEM GUIDE by Ting which now has the assembler in its final chapter. The formation of an Organge County fig group was begun.

Martin Tracy of MicroMotion discussed Implementing Strings in FORTH, their 8th chapter in "FORTH-79 Tutorial and Reference Manual" (for the APPLE II). This string package compares, concatenates, converts and arrays with words like GET$, INPUT$ and IN$ (which indexes into the $tring).

AN OPEN RESPONSE

We continually receive letters asking if FORTH can be installed on a particular computer, particularly those without direct access mass storage or an ASCII terminal (i.e. PET, Vip, and Kim). Often, similar queries reflect a desire to use cassette tape. This summary gives the general characteristics of a system in which FORTH will be responsive. For fig-FORTH installation, an assembler is also needed.

FORTH is an interactive, compiled language. This statement may be expanded to conclude that compilation requires mass storage for source text; it must be random access to be interactive. A terminal is also needed, as a hex keypad cannot be deemed interactive. The character set must be complete for program portability, reflecting the commonality of language.

Requirements to execute:

1. A random access mass storage device with direct access to sector read/write i.e. disk or diskette.

2. 16 Kilobytes of ram.

3. A keyboard input with at least the full upper case ASCII character set.

4. A display of at least 64 characters by 16 lines.

Requirements to install:

1. An assembler that can accept about 80K of source producing about 5.5K of object, either memory or disk.

Requirements derived from the FORTH-79 Standard:

1. 2000 bytes of memory for application dictionary (beyond FORTH, stacks and disk buffers).
2. Stacks of 64 and 48 bytes
3. Mass storage of 32 blocks of 1024 bytes
4. An ASCII terminal

If you are missing any of these elements, we express our condolences. You will have to tolerate an irregular installation and suffer portability problems. This curse is not caused by FORTH but by the shortsightedness of hardware vendors. FORTH is an environment in which you can operate as a professional. We know of no professional who would demand to have his terminal line width reduced to 40 characters, have six ASCII characters removed from his keyboard or return his disk to the manufacturer as unnecessary. If FORTH were compromised to less than the above guidelines, we would ultimately be operating from a hex keypad with paper tape.

BENCHMARKING:

Because there is almost universal disagreement on which are the most valid benchmark tests; and because in FORTH memory compactness may be traded off for execution speed at the implementor's option, it is the policy of FORTH DIMENSIONS to minimize the use of benchmark tests that measure speed alone. Such single dimensional tests more precisely measure the speed of a given CPU than the implementation of FORTH itself and encouraging such simplistic testing will probably mean the compactness of FORTH will inevitably suffer. For these reasons FORTH DIMENSIONS is normally only interested in benchmark tests that measure both productivity (useful work) and speed as a better indicator of a given implementations value.

FORTH VENDORS

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

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

Sierra Computer Co.
617 Mark NE
Albuquerque, NM 87123

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

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

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

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

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

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

M & B Design
820 Sweetbay Drive
Sunnyvale, CA 94086

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

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

PDP-11
Laboratory Software Systems, Inc.
3634 Mandeville Canyon Rd.
Los Angeles, CA 90049
(213) 472-6995

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

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

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

Tom Zimmer
292 Falcato Dr.
Milpitas, CA 95035

6800 & 6809
Talbot Microsystems
5030 Kensington Way
Riverside, CA 92507
(714) 781-0464

TRS-80
Miller Microcomputer Services
61 Lake Shore Rd.
Natick, MA 01760
(617) 653-6136

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

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

6502
Eric C. Rehnke
540 S. Ranch View Circle #61
Anaheim Hills, CA 92087

8080/Z80/CP/M
Laboratory Microsystems
4147 Beethoven St.
Los Angeles, CA 90066
(213) 390-9292

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

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

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

KV33 Corp.
PO Box 27246
Tucson, AZ 85726

68000
Emperical Res. Grp.
PO Box 1176
Milton, WA 98354
(206) 631-4855

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

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

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

Zendex Corp.
6398 Dougherty Rd.
Dublin, CA 94566

Variety of FORTH Products
Interactive Computer Systems, Inc.
6403 Di Marco Rd.
Tampa, FL 33614

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

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

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

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

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

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

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

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

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

NEW PRODUCT ANNOUNCEMENT FORMAT

In the interests of comparison uniformity and completeness of data in new product announcements FORTH DIMENSIONS requests that all future new product announcements use the following format:

1. Vendor name (company)
2. Vendor street address (P.O. Boxes alone are not acceptable for mail order)
3. Vendor mailing address (if different from street address)
4. Vendor area code and telephone number
5. Person to contact
6. Product name
7. Brief description of product use/features
8. List of extras included (editor, assembler, data base, games, etc.)
9. List of machines product runs on
10. Memory requirements
11. Number of pages in manual
12. Tell what manual covers
13. Indicate whether or not manual is available for separate purchase
14. If manual is available indicate separate purchase price and whether or not manual price is credited towards later purchase
15. Form product is shipped in (must be diskette or ROM -- no RAM only or tape systems)
16. Approximate number of product shipments to date (product must have active installations as of writing -- no unreleased products)
17. Product Price
18. What price includes (shipping, tax, etc.)
19. Vendor warranties, post sale support, etc.
20. Order turn around time

HELP WANTED

Openings for a project manager and senior programmer. Both positions offer the opportunity to work on a wide variety of projects, including systems programming and real-time scientific and industrial applications. Salary and benefits are excellent. A starting bonus is available for anyone with a substantial FORTH background. Contact FORTH, Inc.; 2309 Pacific Coast Highway; Hermosa Beach, CA 90254; (213) 372-8493

Programmer analyst to work and live in the Miami area who is trained and experienced in the CYBOS language. Contact Keller Industries, Inc., 18000 State Road 9, Miami, FL 33162; (305) 651-7100, ext. 202.

FORTH programmer for computer graphics. Contact Cornerstone Associates, 479 Winter St., Waltham, MA 02154; (617) 890-3773.

RENEW TODAY!

FORTH, INC. NEWS PAGE

This is the first in a series of columns highlighting various activities at FORTH, Inc.

RECENT APPLICATIONS:

In December Chuck Moore completed work on a 24-channel video mixer for Homer & Associates, a producer of films for promotioal and entertainment purposes in Hollywood. This Z-80 based system controls 16 slide projectors, four movie projectors and audio tape. It has mastering and sequencing capabilities which Peter Conn, Homer's president, says are unique in the industry.

In early January American Airlines performed the final acceptance of the LAX outbound baggage system developed by Dean Sanderson and Mike LaManna. The system runs on two PDP-11 computers (one functions as a backup), and controls several conveyor belts, bag encoder stations, electric eye sensors, and printers. It is more accurate and has 25% better performance than the all-assembly language system it replaced.

NEW PRODUCTS:

EXORset polyFORTH pF6809/30, developed by Mike LaManna, is our newest product and runs on the Motorola EXORset 30 -- a microcomputer featuring a 6809 processor, graphics CRT and two mini-floppies in a single compact box. EXORset polyFORTH sells for $4750 and incudes a secial screen editor; a high-speed graphics option with software vector and character generation; labeled graphs with several plotting modes; a "strip-chart" function with snap-shot capabilities, and several demonstration routines. EXORset polyFORTH sells for $4750.00. The option package sells for $500.00. Both will be featured at FORTH, Inc.'s spring seminar series.

FORTH - 79:

Al Krever is working on a new release of polyFORTH scheduled for March. This new release will feature many improvements in all systems, plus greater compatability with the FORTH - 79 Standard.

The FORTH - 79 edition of USING FORTH has been sent to the printers and will be available after mid-February.

POLYFORTH COURSES:

FORTH, Inc. offers two courses--an introductory course for programmers unfamiliar with polyFORTH and an advanced course designed for those with considerable FORTH experience who desire greater familiarity with system level functions, target compiling and other advanced techniques.

FORTH, Inc.'s course schedule for the next few months is:

Month	Introductory	Advanced
April	6 - 10	13 - 17
May	11 - 15 (tentative)	

Contact Carol Ritscher at FORTH, Inc. (213) 372-8493 for more information.

SEMINARS & WORKSHOPS:

A series of completely new half-day seminars and one-day workshops has been scheduled in several cities. Both present an overview of the features and benefits of polyFORTH for professional users. The EXORset and tis new graphics package will be featured.

City	Seminar	Workshop
Washington, DC	3/19	3/20
Houston	4/21	4/22
Boston	4/23	4/24

Contact Carol Ritscher at FORTH, Inc. (213) 372-8493 for more information.

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 Saturday	FIG Monthly Meeting, 1:00 p.m., at Southland Shopping Ctr., Hayward, CA. FORML Workshop at 10:00 a.m.
Southern California 4th Saturday	FIG Meeting, 11:00 a.m. Allstate Savings, 8800 So. Sepulveda, L.A. Call Phillip Wasson, (213) 649-1428.
Massachusetts 3rd Wednesday	MMSFORTH Users Group, 7:00 p.m., Cochituate, MA. Call Dick Miller at (617) 653-6136 for site.
San Diego Thursdays	FIG Meeting, 12:00 noon. Call Guy Kelly at (714) 268-3100 x 4784 for site.
Seattle	Chuck Pliske or Dwight Vandenburg at (206) 542-8370.
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) 729-3320 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
Oregon	Ed Krammerer at (503) 644-2688.
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, President, ASR Corp. Int'l, 3-15-8, Nishi-Shimbashi Manato-ku, Tokyo, Japan.
Quebec, Canada	Gilles Paillard at (418) 871-1960.

Publishers Note:

Please send notes (and reports) about your meetings.

RENEW

RENEW TODAY!