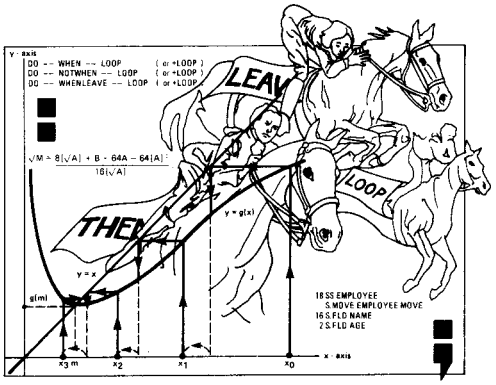
FORTH Dimensions

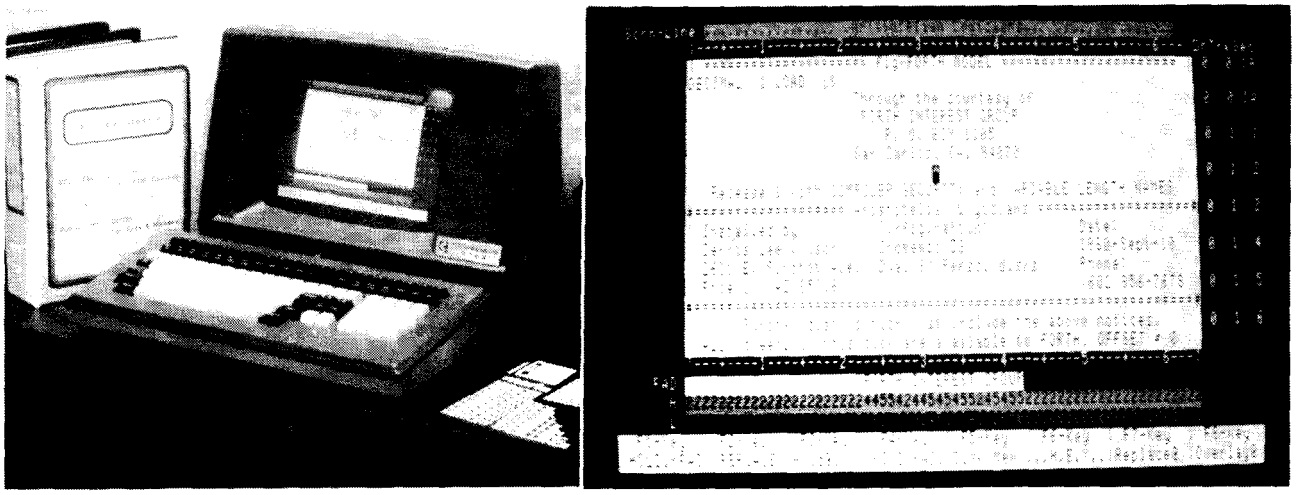
Volume 5, Number 6
March/April 1984
$2.50

PL/I Data Structures



FEATURES

DEPARTMENTS



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

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

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

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

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

	USA	Foreign AIR
☐ **FIG-FORTH & Full Screen EDITOR package**		
Minimum system requirements:		
80x24 video screen w/ cursor addressability		
8080 or Z80 or compatible cpu		
CP/M or compatible operating system w/ 32K or more user RAM		
Select disk format below, (soft sectored only)	$50	$65
☐ 8" SSSD for CP/M (Single Side, Single Density)		
Cromemco CDOS formats, Single Side, S/D Density		
☐ 8" SSSD ☐ 8" SSDD ☐ 5¼" SSSD ☐ 5¼" SSDD		
Cromemco CDOS formats, Double Side, S/D Density		
☐ 8" DSSD ☐ 8" DSDD ☐ 5¼" DSSD ☐ 5¼" DSDD		
Other formats are being considered, tell us your needs.		
☐ Printed Z80 Assembly listing w/ xref (Zilog mnemonics)	$15	$18
☐ Printed 8080 Assembly listing	$15	$18
TOTAL $_____		_____

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

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

FORTH Dimensions
Published by Forth Interest Group
Volume V, No. 6
March/April 1984
Editor
Marlin Ouverson
Publisher
Roy C. Martens
Typesetting/Production
LARC Computing, Inc.
Cover Art
Al McCahon

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

Letters to the Editor

Appeal to Vendors

Dear FIG,

As a neophyte Forth enthusiast and typical *Compute!*-reading consumer, I would like to use this letter to tell Forth vendors what I think they ought to produce for the mass market. First of all, I think we all owe an enormous debt to Laxen and Perry for their F83 model, which I'm sure most of you have a copy of. The **VIEW** feature alone puts it in a class by itself. Why not use it as the basis of a really deluxe Forth system? Obviously, source code is essential for such a system; but who wants to program in Forth without source code? I'm willing to pay extra for it, but I think it must be available. The following is a list of enhancements to F83 that, if put into a nice package with good documentation and support, could sell all day long at $250 per copy.

- A decent screen editor.
- Improved shadow-screen documentation, including high-level source for code words. Include two shadow screens per source screen, if necessary. Ideally, this would just about eliminate the need for a big manual.
- Automatic inclusion of new files in **VIEW**.
- Toggle to automatically drop into the editor if an error occurs while compiling a screen.
- Separate heads and bodies. (An earlier version of the F83 meta-compiler had this feature.)
- The ability to generate ROMable code. Ideally, the meta-compiler would put the bodies of only the words actually used into a file that could be read like a .COM file in CP/M.
- An optimizer feature like Auto-Opt from Harvard Softworks or the Native Code Compiler from Labora-

Editorial
Forth on Tap

We are pleased to announce a new department to appear regularly in our pages. "Ask the Doctor" is addressed specifically to readers' needs. If you are looking for an answer and can't seem to find it, or don't understand a fine point of Forth programming, write a short letter explaining your problem. This is an especially good opportunity for some of you newer Forth folks who have questions. So write!

The new column is being handled by FIG President Bill Ragsdale. If you haven't yet made his acquaintance, turn to the interview with him in this issue — you'll also learn about FIG's origin, the evolution of Forth and Bill's insights into the Forth marketplace.

Do you ever find yourself modifying perfectly good code just because your fingers are itchy? Maybe it's only "keyboard withdrawal." Instead of infinite revision, why not write an article? Or if you are stuck for good ideas, ring up one of the on-line Forth conferences. Arpanet carries a list of people interested in and/or using Forth. The list currently operates as a "delayed distribution, non-digestified" mailing list of people to whom accumulated messages are sent. To send messages to the list, address them to:

FORTH@SRI-CSL or
INFO-FORTH@SRI-CSL.

To get added to the list, send a note to FORTH-REQUEST@SRI-CSL. And don't forget that FIG has its own CBBS system: call 415-538-3580, answer a couple of simple questions and type "Read conferences" to see what subject areas are in progress. There is lots of help, and lots of categories like software, Forth standards, humor, vendors, enquiries, marketplace, etc. Give it a try; you're sure to find something useful.

Meanwhile, if you are reading this at the West Coast Computer Faire, drop by the FIG booth and say hello. Or attend one of the lectures and demonstrations. And since this is the last issue in the current volume of *Forth Dimensions*, it's time to renew your membership! Do it in person at the Faire or mail the reply envelope included this month, but do it soon — you won't want to miss the coming issues!

—Marlin Ouverson
Editor

SEND A CHECK TO FIG TODAY!
MAKE THIS YOUR BEGINNING!
RENEW NOW!

tory Microsystems. One would like to be able either to code a single word or to have the program find that mystical five percent of the source which really slows the application down, and have those words optimized automatically.

• Pre-compiled (and pre-optimized?) source code which could be loaded from a screen.

• Perhaps even a "library" file of pre-compiled blocks which could be loaded by a high-level word name. This would be the equivalent of procedures or functions in C and Pascal. The idea is to get these things loaded automatically from a different file without having to leave **FORTH.BLK** (or whatever) and open another file. Of course, hard-core Forth freaks would think this is silly, but as a typical consumer, I am not yet willing to abandon CP/M and MS-DOS.

With these enhancement to F83, I think the average tinkerer like me has an ideal (and commercially attractive) environment for developing software. As one who would rather buy VisiCalc than re-write it, but who occasionally has the need to write programs, I place a premium on development time; and that, of course, is one of the areas in which Forth excels. I think this should be stressed more in advertising. With the system outlined above, one would scarcely need to take his eyes off the screen to write a small program. And with optimized, ROMable code, who could complain about inefficiency?

Sincerely,

Alan Huth
1828A Diamond
San Diego, CA 92109

VIC Dump

Dear Sirs:

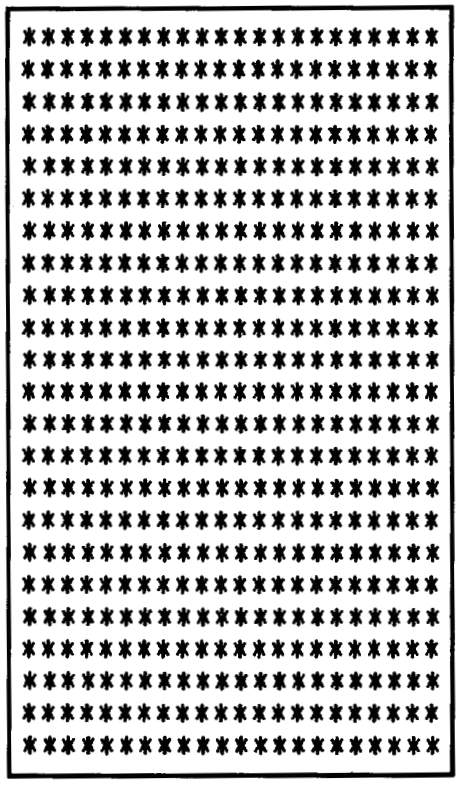
I am a novice Forth enthusiast using a system that many consider a toy: the VIC-20 by Commodore. I have had my computer for almost a year now, and have become proficient in BASIC and Pilot, and am adequate in Assembler.

I have to say that since I got my Forth cartridge (from HES) I have done little or no programming in anything but Forth. The language is not only elegant and versatile, it's also lots of fun. My first project was converting BREAKFORTH (*BYTE*, August 1980) to run on my computer.

Since then, I have implemented the data base in Leo Brodie's book, *Starting Forth* on my VIC and have developed an artificial intelligence demonstration program based on an algorithm in David Heiserman's *Projects in Machine Intelligence for Your Home Computer.*

I've included in this letter some code that dumps the screen to a printer with a little demo word included. This definition is, unfortunately, machine specific because it converts Commodore screen codes to ASCII by PEEKing each screen location. It would be easy to convert this to a CBM-64 and, I think, to most other PCs.

```
SCR # 1
 0  ( FORTH SCREEN DUMP   M. MARSH   12/8/83   STA. BARBARA, CA. )
 1
 2
 3  : SCR.DUMP   PRON 30 SPACES 506 0
 4                    DO I 4096 + C@
 5                         DUP 31 < IF 64 + EMIT ELSE
 6                         DUP 63 > IF 32 + EMIT ELSE
 7                         DUP 31 > IF      EMIT ELSE
 8                         DROP     ENDIF ENDIF ENDIF
 9                       I 1 + 22 MOD 0 = IF CR 30 SPACES ENDIF
10                    LOOP
11               CLALL ;
12
13
14  : DEMO   4096 506 42 FILL SCR.DUMP ;
15
```



Now for a question: in Dr. Turpin's recursive stack sort (*Forth Dimensions* V/2), what is the definition of **MYSELF** (I can guess the function)? I could not find this word anywhere, though I have the feeling I've read it somewhere.

If there are any VIC Forthers out there, **DROP** me a line and we'll **SWAP** ideas.

Sincerely,

Michael Marsh
6506 Sabado Tarde
Isla Vista, CA 93117

As for MYSELF...

Gentlemen:

I am a new member of FIG and have just received my first issues of *Forth Dimensions*. I like it well enough to order the two previous volumes, but the first routine I tried to use was the "Recursive Sort on the Stack" (*Forth Dimensions* V/2). On the fourth line of the **SINK** definition appears the word **MYSELF**. I have no idea what this does or whether it is a standard word in another Forth version. I know I do not have it. (I am using Laboratory Microsystems Z80 Forth.)

I would like to echo the lament of Dave Kuhlman where he complains about undefined words. I am, like Dave, new to Forth and I am sure there are many more like us, the way Forth is spreading. We need clearly defined words because, as he says, we cannot be expected to be able to fill in the gaps.

Sincerely,

William T. Watson
Route 3, Box 46
Theodosia, MO 65761

Dear Marlin:

With reference to your letter, the definition of **MYSELF** *is:*

: MYSELF LATEST PFA CFA , ; IMMEDIATE

This definition is derived from page seventy-seven of the SYM-FORTH Source Code Listing.

Sincerely,

Richard H. Turpin, Ph.D.

(Continued on page 12)

Ask the Doctor

Bill Ragsdale
Hayward, California

Throughout the history of the Forth Interest Group, we have gotten letters of inquiry on the application and learning of Forth. When possible, personal answers were sent. This new column intends to broaden users' questions into a forum of general interest. The Doctor solicits your questions about Forth for future columns. Specialists will be called in for consultation on vendor-specific questions. Just address your question to: Ask the Doctor, P.O. Box 1105, San Carlos, CA 94070.

This month, the good Doctor answers the question: "I always seem to use **COUNT** just before **TYPE**. If this is the only use, why shouldn't **TYPE** include the **COUNT** function?"

Answer: Forth generally keeps word definitions short and generalized. You have a large number of simple commands at your disposal and the word **COUNT** is a good example of this generality. It accepts the address in memory of a text string which begins with a character-count byte. **COUNT** fetches this count and advances the address by one to the first character of the sequence. The FIG Model Glossary and Forth-83 define the word in these terms and suggest just the typical use with **TYPE**. If **TYPE** included the counting function, then you would need another version of **TYPE** to handle the cases in which the count byte did not precede the text in memory.

But more uses are available than just with **TYPE**.

Definition of COUNT

We had first better examine **COUNT** itself. Its fig-FORTH definition is:

```
: COUNT DUP 1+ SWAP C@ ;
```

A definition more obvious in action is:

```
: COUNT 1+ DUP 1- C@ ;
```

```
Blk 12
0 ( Examples for COUNT                                  83feb13  WFR )
1 ( options to update fig-FORTH words to Forth 83 )
2 : CREATE    0  VARIABLE  -2  ALLOT  ;
3 : WORD      WORD  HERE ;
4
5
6 ( Example #1 )
7 DECIMAL  34  CONSTANT  QUOTES  32  CONSTANT  BL
8
9 : MAKE    CREATE  QUOTES  WORD  C@  1+  ALLOT  ;
10
11 MAKE  DEMO  This is demo text"
12
13
14
15


Blk 13
0 ( Examples for COUNT                                  83feb13  WFR )
1
2 ( Example #2 )
3 : DAYS:     ( compile n day names )
4      0  DO   BL  WORD  C@  1+  ALLOT   LOOP  ;
5
6 CREATE  WEEK
7 7 DAYS:    SUNDAY  MONDAY  TUESDAY
8            WEDNESDAY  THURSDAY  FRIDAY  SATURDAY
9
10 : STEP     ( 0..6 --- addr )
11      ?DUP  IF   0  DO  COUNT  +  LOOP   THEN  ;
12
13 : DAY      ( day# --- address )
14       1-  0  MIN  6  MAX    WEEK  SWAP  STEP  COUNT  TYPE ;
15
```

```
Blk 14
0 ( Examples for COUNT                                  83feb13  WFR )
1
2 ( Example #3 )
3 :  BOX       CREATE   C,  C,  C,  ;
4 :  SHAPE     ( address --- )
5     COUNT  .    COUNT  .    C@  ;
6
7     1  1  10  BOX  ROD
8     3  3  3  BOX  CUBE
9     1  2  3  BOX  PARALLELEPIPED
10
11 ( Example #4 )
12 CREATE WORKSPACE     31  ALLOT
13 :  FRESH   ( clear the buffer to blanks )
14     30  WORKSPACE  C!  ( Store empty count )
15     WORKSPACE  COUNT  BL  FILL  ;
```

The latter definition increments the address, duplicates it, backs up and fetches the count byte from memory. If your system is still fig-FORTH you will have to update a couple of words to Forth-83. Just load the corrections given in Block #12 of the figures.

Counted Text

Example one consists of a word to make named string literals and display them. The word **MAKE** creates a new word in the dictionary and adds the following text ending in double quote marks. After loading from Block #12, just type **DEMO COUNT TYPE** to see the string printed. From the address left by **DEMO**, **COUNT** converts to the start of text and the character count, just as expected by **TYPE**. This is the usual use of **COUNT**.

Ending Addresses

Another use of **COUNT** is to find the ending address of a string (with count). This is done by adding the address and character count together. Example two places the days of the week sequentially in memory. You may then request any day by its number. The days are numbered 1 through 7 starting with Sunday.

In the word **STEP** we are using **COUNT** in the phrase **DO COUNT + LOOP** to get the character count of the string and add this count to the address of the first character. This steps to the start of the next string. The **?DUP IF . . . THEN** structure hops over this loop if we want the address of the first (zero-th) day.

We conclude **DAY** using **COUNT** in its usual form, to get the count of the day name we wish to display. You may test **DAY** by typing: **5 DAY** and seeing THURSDAY output.

Scanning Bytes

Notice that **COUNT** fetches a byte from memory and increments to the next address. How convenient if we wish to fetch a sequence of bytes! Just use the fetched byte and **COUNT** again. This construct is useful in Forth assemblers, which often use several byte parameters in sequence.

Our example three places in memory the dimensions we assign to several geometrical figures. **BOX** creates the dictionary entry followed by three values. The word **SHAPE** prints the shape of the box, three dimensions. To test, type **ROD SIZE**. From the address left by **ROD** the word **SIZE** will fetch the first dimension byte and print it; from the incremented address it will fetch the second dimension byte and print it; and finally it will **C@** the last dimension. Note that **C@** is sufficient, as we no longer need to maintain the address.

Manipulating Memory

COUNT can be used in character storage even if text is not present; we only need the count. Example four creates a text workspace and **FRESH**ens it up with blanks. Here we have a thirty-character storage area named **WORKSPACE** with room for the count. **FRESH** first shoves the maximum character count into the first byte of **WORKSPACE**. **COUNT** then gets the first character address and the count, which are needed by **FILL** (which fills with blanks). Other words count fill-in text and manipulate within the **WORKSPACE**.

Now that the good Doctor has opened your vistas on the word **COUNT**, please take the opportunity to write with your own questions on usage, Forth internals or allied problems. We'll bring in specialists as needed.

THIS IS THE END!

THE END OF YOUR MEMBERSHIP?
DON'T LET IT HAPPEN!
RENEW TODAY!

PL/I Data Structures in Forth

Bruce W. Walker
San Pedro, California

This paper gives a simple way of adding data structures (as in PL/I, COBOL or Pascal) to Forth. It has three sections: first, what data structures are and how they can be used in Forth (the cookbook approach); second, how the words work; and finally, the words themselves.

Data Structures

Forth programs usually make do with just two data types: numbers and arrays of numbers. While this is adequate for scientific programming, commercial programmers have known since time immemorial (*i.e.* before 1960) that many programs are easier to design and write using the concept of a data structure. A data structure is a heterogeneous collection of data, and can be thought of either as a single entity or as a collection with named parts, whichever is more convenient at the time.

While Forth has little data structure machinery built in, it does have the tools needed to add new data types. Other authors have worked on user stacks[5], complex numbers[2], quad precision[1] and character strings[3,6]. The closest thing I have seen to data structure capability in Forth is in the article on data base design[4]; that article is a gold mine of ideas and should be read by serious Forth programmers once a year.

Figure One

```
DECLARE   1 EMPLOYEE,
            2 NAME CHARACTER (16),
            2 AGE FIXED BINARY (15,0);
```

In PL/I, a program can have a declaration like in figure one. This means that **EMPLOYEE** has two parts, **NAME** and **AGE**. The fields can be referenced individually as **EMPLOYEE.NAME** and **EMPLOYEE.AGE**. The advantage over two separate variables is that **EMPLOYEE** can be used in assignment as a

```
                    FIG FORTH VERSION                                   00000100
SCR # 110                                                               00000200
 0 ( PL/I STRUCTURES )                                                  00000300
 1 ( TOTAL LENGTH, CURRENT LENGTH, FIELD LENGTH )                       00000400
 2 0 VARIABLE TLEN                                                      00000500
 3 0 VARIABLE CLEN                                                      00000600
 4 0 VARIABLE FLEN                                                      00000700
 5 ( INITIALIZE STRUCTURE VARIABLES )                                   00000800
 6 : INIT.SV TLEN ! 0 CLEN ! ;                                          00000900
 7                                                                      00001000
 8 ( STORE STRUCTURE VARIABLES )                                        00001100
 9 : STR.SV CLEN @ FLEN @ - , FLEN @ , ;                                00001200
10                                                                      00001300
11 -->                                                                  00001400
12                                                                      00001500
13                                                                      00001600
14                                                                      00001700
15                                                                      00001800
                                                                        00001900
SCR # 111                                                               00002000
 0 ( ALLOCATE STATIC STRUCTURE )                                        00002100
 1 : SS ( SIZE --- )                                                    00002200
 2 <BUILDS DUP ALLOT INIT.SV DOES> ;                                    00002300
 3 ( BEGIN FIELD DEFINITION FOR DYNAMIC STRUCTURES )                    00002400
 4 : DS ( SIZE --- ) INIT.SV ;                                          00002500
 5 ( ALLOCATE AN ARRAY WITH A GIVEN ELEMENT SIZE )                      00002600
 6 : S.ARRAY <BUILDS , ALLOT DOES>                                      00002700
 7 DUP 2+ >R @ * R> + ;                                                 00002800
 8 ( MOVE A WHOLE STRUCTURE )                                           00002900
 9 : S.MOVE <BUILDS TLEN @ , DOES>                                      00003000
10 @ CMOVE ;                                                            00003100
11 -->                                                                  00003200
12                                                                      00003300
13                                                                      00003400
14                                                                      00003500
15                                                                      00003600
SCR # 112                                                               00003700
 0 ( CREATE A FIELD    SIZE ---                                         00003800
 1   AT RUN TIME                                                        00003900
 2   STRUCTURE-ADDRESS --- FIELD ADDRESS )                              00004000
 3 : S.FLD <BUILDS CLEN @ , DUP FLEN ! CLEN +! DOES>                    00004100
 4   @ + ;                                                              00004200
 5 ( MOVE FROM AN ADDRESS TO A FIELD )                                  00004300
 6 : FLD.MA <BUILDS STR.SV DOES>                                        00004400
 7   DUP 2+ >R @ + R> @ CMOVE ;                                         00004500
 8 ( MOVE FROM A FIELD TO AN ADDRESS )                                  00004600
 9 : FLD.MB <BUILDS STR.SV DOES>                                        00004700
10   >R SWAP R @ + SWAP R> 2+ @ CMOVE ;                                 00004800
11                                                                      00004900
12 -->                                                                  00005000
13                                                                      00005100
14                                                                      00005200
15                                                                      00005300
SCR # 113                                                               00005400
 0 ( MOVE A FIELD BETWEEN STRUCTURES )                                  00005500
 1 : FLD.MC <BUILDS STR.SV DOES>                                        00005600
 2   >R R @ + SWAP R @ + SWAP R> 2+ @ CMOVE ;                           00005700
 3 ( MOVE AN ITEM OF A FIELD LENGTH )                                   00005800
 4 : FLD.MD <BUILDS FLEN @ , DOES>                                      00005900
 5   @ CMOVE ;                                                          00006000
 6 ( PREPARE TO DEFINE SUBFIELDS )                                      00006100
 7 : SF CLEN ! ;                                                        00006200
 8                                                                      00006300
 9 ;S                                                                   00006400
10                                                                      00006500
11                                                                      00006600
12                                                                      00006700
13                                                                      00006800
14                                                                      00006900
15                                                                      00007000
```

```
                    79 STANDARD VERSION                          00007100
SCR # 110                                                        00007200
 0 ( PL/I STRUCTURES )                                           00007300
 1 ( TOTAL LENGTH, CURRENT LENGTH, FIELD LENGTH )                00007400
 2 0 VARIABLE TLEN                                               00007500
 3 0 VARIABLE CLEN                                               00007600
 4 0 VARIABLE FLEN                                               00007700
 5 ( INITIALIZE STRUCTURE VARIABLES )                            00007800
 6 : INIT.SV TLEN ! 0 CLEN ! ;                                   00007900
 7                                                               00008000
 8 ( STORE STRUCTURE VARIABLES )                                 00008100
 9 : STR.SV CLEN @ FLEN @ - , FLEN @ , ;                         00008200
10                                                               00008300
11 -->                                                           00008400
12                                                               00008500
13                                                               00008600
14                                                               00008700
15                                                               00008800
                                                                 00008900
SCR # 111                                                        00009000
 0 ( ALLOCATE STATIC STRUCTURE )                                 00009100
 1 : SS ( SIZE --- )                                             00009200
 2 CREATE DUP ALLOT INIT.SV DOES> ;                              00009300
 3 ( BEGIN FIELD DEFINITION FOR DYNAMIC STRUCTURES )             00009400
 4 : DS ( SIZE --- ) INIT.SV ;                                   00009500
 5 ( ALLOCATE AN ARRAY WITH A GIVEN ELEMENT SIZE )               00009600
 6 : S.ARRAY CREATE , ALLOT DOES>                                00009700
 7 DUP 2+ >R @ * R> + ;                                          00009800
 8 ( MOVE A WHOLE STRUCTURE )                                    00009900
 9 : S.MOVE CREATE TLEN @ , DOES>                                00010000
10 @ CMOVE ;                                                     00010100
11 -->                                                           00010200
12                                                               00010300
13                                                               00010400
14                                                               00010500
15                                                               00010600
SCR # 112                                                        00010700
 0 ( CREATE A FIELD    SIZE ---                                  00010800
 1   AT RUN TIME                                                 00010900
 2   STRUCTURE-ADDRESS --- FIELD ADDRESS )                       00011000
 3 : S.FLD CREATE CLEN @ , DUP FLEN ! CLEN +! DOES>              00011100
 4   @ + ;                                                       00011200
 5 ( MOVE FROM AN ADDRESS TO A FIELD )                           00011300
 6 : FLD.MA CREATE STR.SV DOES>                                  00011400
 7   DUP 2+ >R @ + R> @ CMOVE ;                                  00011500
 8 ( MOVE FROM A FIELD TO AN ADDRESS )                           00011600
 9 : FLD.MB CREATE STR.SV DOES>                                  00011700
10   >R SWAP R@ @ + SWAP R> 2+ @ CMOVE ;                         00011800
11                                                               00011900
12 -->                                                           00012000
13                                                               00012100
14                                                               00012200
15                                                               00012300
SCR # 113                                                        00012400
 0 ( MOVE A FIELD BETWEEN STRUCTURES )                           00012500
 1 : FLD.MC CREATE STR.SV DOES>                                  00012600
 2   >R R@ @ + SWAP R@ @ + SWAP R> 2+ @ CMOVE ;                  00012700
 3 ( MOVE AN ITEM OF A FIELD LENGTH )                            00012800
 4 : FLD.MD CREATE FLEN @ , DOES>                                00012900
 5   @ CMOVE ;                                                   00013000
 6 ( PREPARE TO DEFINE SUBFIELDS )                               00013100
 7 : SF CLEN ! ;                                                 00013200
 8                                                               00013300
 9 ;S                                                            00013400
10                                                               00013500
11                                                               00013600
12                                                               00013700
13                                                               00013800
14                                                               00013900
15                                                               00014000
```

Figure Two

```
DECLARE   1 BOSS,
            2 NAME CHARACTER(16),
            2 AGE FIXED BINARY (15,0);
```

whole. For example, suppose we also have the declaration in figure two. Then the following assigns both fields:
BOSS = EMPLOYEE;

To do the same in Forth, we need to declare the structure, its fields, and create routines to do the assignment. These turn out to be fairly easy with **CREATE** and **DOES>**. The words are given in the concluding section of this article. The above example of a structure, in Forth, would look like figure three. (I have not included all the move

Figure Three

```
18   SS EMPLOYEE
     S.MOVE EMPLOYEE.MOVE
16   S.FLD NAME
 2   S.FLD AGE
```

functions, which are optional.) References to the field addresses are then:

EMPLOYEE NAME

EMPLOYEE AGE

And the group assignment is given by:

EMPLOYEE BOSS EMPLOYEE.MOVE

(Note that you don't re-declare name and age for **BOSS**.)

Two generalizations of data structuring also appear in PL/I. First, the elements can be arrays, as in figure four.

Figure Four

```
DECLARE   1 EMPLOYEE(100),
            2 NAME CHARACTER(16),
            2 AGE FIXED BINARY (15,0);
```

This declares an array of data structures, here 100 employees. A program can then have both the statements shown in figure five.

Figure Five

```
EMPLOYEE(M).NAME = EMPLOYEE(N).NAME;
```

To do this in Forth, we create an array of elements and use the dynamic allocation form of structure declaration, as in figure six. Figure seven is, then, the Forth version of figure five.

Figure Six

```
1800   18 S.ARRAY EMPLOYEE
          18 DS
             S.MOVE EMPLOYEE.MOVE
          16 S.FLD NAME
             FLD.MC NAME.MOVE
           2 S.FLD AGE
```

Figure Seven

```
N @ EMPLOYEE M @ EMPLOYEE
EMPLOYEE.MOVE
              and
N @ EMPLOYEE M @ EMPLOYEE NAME.MOVE
```

The second generalization is to have more than one level, as in figure eight. Now a reference to an individual field looks like:

EMPLOYEE(N).NAME.INITIALS

Figure Eight

```
DECLARE   1 EMPLOYEE(100),
            2 NAME,
              3 INITIALS CHARACTER(2),
              3 LASTNAME CHARACTER(14),
            2 AGE FIXED BINARY (15,0);
```

In Forth, you can do the same thing by re-starting the count for the subfield by declaring the whole main structure first, then by declaring the subfield, as in figure nine.

Figure Nine

```
1800     18 S ARRAY EMPLOYEE
            18 DS
               S.MOVE EMPLOYEE.MOVE
            16 S.FLD NAME
               FLD.MC NAME.MOVE
             2 S.FLD AGE
   0           NAME SF
             2 S.FLD INITIALS
            14 S.FLD LASTNAME
```

So, in Forth the sub-field reference would be:

N @ EMPLOYEE INITIALS

A typical application of a data structure is a table with two fields: KY, the key field, and VAL, the value field. The table must be kept sorted on the key field when new entries are added.

```
                                                                    00014100
                TRACING COLON DEFINITIONS                           00014200
SCR # 1                                                             00014210
 0 1 VARIABLE TRACE                                                 00014300
 1 : OUTX ( OUTPUT A NUMBER IN HEX, WHILE SAVING OLD BASE )         00014400
 2   BASE @ HEX SWAP 6 .R SPACE BASE ! ;                            00014500
 3                                                                  00014600
 4 : TRACE.SUB ( CREATE A TRACE LINE AND CHECK FOR USER INTERRUPT )  00014700
 5   CR SP@ OUTX SP@ 2 - @ OUTX R OUTX R 4 - @ 2+ NFA ID.           00014800
 6   ?TERMINAL IF KEY 27 = IF ABORT THEN THEN ;                     00014900
 7                                                                  00015000
 8 : NINTERPRET BEGIN -FIND                                         00015100
 9   IF STATE @ <                                                   00015200
10      IF CFA , TRACE @ ( TRACING TURNED ON? )                     00015300
11        IF ' TRACE.SUB CFA , ( STORE TRACE ROUTINE CFA ) THEN     00015400
12      ELSE CFA EXECUTE THEN ?STACK                                00015500
13   ELSE HERE NUMBER DPL @ 1+                                      00015600
14   IF  COMPILE  DLITERAL ELSE DROP  COMPILE  LITERAL THEN ?STACK  00015700
15   THEN AGAIN ; -->                                               00015800
SCR # 2                                                             00015900
 0                                                                  00016000
 1 : NQUIT ( RUN WITH TRACING )                                     00016100
 2 0 BLK !  COMPILE                                                 00016200
 3 BEGIN RP! CR QUERY NINTERPRET                                    00016300
 4 STATE @ 0= IF ." OK" THEN                                        00016400
 5 AGAIN ;                                                          00016500
 6 ;S                                                               00016600
 7                                                                  00016700
 8                                                                  00016800
 9                                                                  00016900
10                                                                  00017000
11                                                                  00017100
12                                                                  00017200
13                                                                  00017300
14                                                                  00017400
15                                                                  00017500
```

This means that when adding a new entry, part of the processing will involve only the key field (namely, doing the test to find where the new entry should go), while the other part of the processing involves both fields (namely, moving the other entries out of the way).

The table in figure ten has a counter entry **ACT** which gives the actual number of entries in use. The body of the table has two fields, **KY** and **VAL**, both sixteen-bit quantities. **#E** is the number of entries and has been made into a named constant on general good programming grounds. A convention is made that the table will have a dummy entry with the maximum possible key value, at all times. This simplifies the search, as when adding a new entry you can be sure that you will always eventually come to a larger key field.

TBL.INIT initializes the table, including putting the dummy entry in the first position, so that the condition assumed by the search routine does hold. **TBL.F** assumes that there is an entry to be added on the stack, and compares its key field with the key fields of the entries in the table in turn, until an entry with a greater key field is found. **TBL.F** then leaves that index on the stack. **TBL.I** assumes that there is an index on the stack and moves all entries after that down one entry. Finally, the user word **TBL.INSERT** actually oversees the whole operation. It checks to see if there is room in the table, returning 1 if not. If there is, it uses **TBL.F** to find the index where the entry should go. It then uses the index twice, once as an argument to **TBL.I** to clear out the space, and once as an argument to **TBL.MOVE** to put the new value in place. Finally, it returns 0 to indicate that the entry was successfully put in the table.

Figure Ten

```
1 VARIABLE ACT

25 CONSTANT #E

#E 4 * 4 S.ARRAY TBL
4 DS  S.MOVE  TBL.MOVE
2 S.FLD KY   2 S.FLD VAL

: TBL.F ACT @ 0 DO
     DUP KY @ I TBL KY @ <
     IF DROP I LEAVE THEN LOOP  ;

: TBL.I 1 - #E 1 - DO
     I TBL I 1 + TBL TBL.MOVE -1 +LOOP  ;

: TBL.INSERT ACT @ #E = IF DROP 1 ELSE
     DUP TBL.F DUP TBL.I TBL
     TBL.MOVE  0  THEN  ;

: TBL.INIT 32767 0 TBL KY ! 1 ACT  !  ;
```

How the Functions Work

As with all **CREATE...DOES>** words, there is action at definition time as well as execution time. In this case, the action at definition time includes saving some information for other words which may be compiled later, as well as storing into the dictionary. All the words use three variables for saving information at definition time. They are:

TLEN total length of the data structure

CLEN length of the data structure up to the field currently being worked on

FLEN length of the field currently being worked on

These variables are used to compute the data saved by the various move routines, data which is used by the (common) **DOES>** part of the routines. The simplest example is **S.MOVE**, move a whole structure. When an operation like

```
S.MOVE  EMPLOYEE.MOVE
```

is executed, the Forth compiler uses **TLEN @ ,** to put the current value of **TLEN** in **EMPLOYEE.MOVE**. When **EMPLOYEE.MOVE** is executed, the address of **EMPLOYEE.MOVE** is put on the stack and the **DOES>** part of **S.MOVE** is executed. This fetches the value at that location, which will be the value of **TLEN** when **S.MOVE** was executed, and then does a **CMOVE** which moves the whole structure (as advertised).

There are a few things to note about the use of the individual words. You would use **SS** to allocate a static structure and begin the data structure machinery, only when you want both functions of allocation and data structure. When the structure is already allocated, in one way or another, you just use **DS** to start a dynamic structure. (The other ways that the data structure might already be allocated are if it is in an array created with **S.ARRAY** or if it is a re-start to create a sub-field.)

It seems unnecessary to have four routines to move fields around (between two similar structures, from a field to an arbitrary address, from an arbitrary address to a field, between two arbitrary addresses), but all the functions are needed in some place or another; I couldn't think of any clever names for the four cases, either. The words would have been somewhat simpler using parameter passing[7], but this would also have slowed down the words somewhat, and field references are likely to be heavily used.

Detailed Descriptions of the Words

Since many of the words have both a compile-time action and a run-time action, each description has up to three parts: first, the dictionary entry built by the word; second, the compile-time action of the word; third, the run-time action of the word. For the more complex words, there is a play-by-play description of the operation of the word, along with descriptions of the stack, enclosed in brackets.

INIT.SV (run time) save the total length of the structure in **TLEN**, initialize the pointer to the start of the current allocated field **CLEN** to zero

STR.SV (run time) save offset of this field and field length; this is called by compile-time words and hence creates a dictionary entry.
(dictionary entry)
word one = field offset
word two = field length

SS (run time) allocate an array, then invoke **INIT.SV**

DS (run time) synonym for **INIT.SV**

S.ARRAY (compile time) save the element size, then allocate an array:
(dictionary entry)
word one = element size
word two = array entries
(run time) multiply element size by element index, and add base

S.MOVE (compile time) save total length:
(dictionary entry)
word one = total length
(run time) get saved length and use **CMOVE** — the 'from' and 'to' address must already be on the stack

S.FLD (compile time) save start offset of this field, store field length in **FLEN** and add to start address pointer:
(dictionary entry)
word one = starting offset
(run time) get start address of field, and add to base address

FLD.MA (compile time) invoke **STR.SV** to save characteristics of this field:
(dictionary entry)
word one = field offset
word two = field length
(run time)
[address data-structure-address dictionary-ptr]
DUP dup address of dictionary entry
2+ point at address of field length
>R save field length address on return stack
@ get field offset
+ add field offset to data structure base
R> get field length address
@ get field length
[address data-structure-field-address length]
CMOVE move from address to data structure

FLD.MB (compile time) invoke **STR.SV** to save characteristics of this field:
(dictionary entry)
word one = field offset
word two = field length
(run time)
[data-structure-address address dictionary-ptr]
>R save dictionary pointer
SWAP swap data structure addresses
R get dictionary pointer
@ get field offset
+ add to data structure address
SWAP swap data structure addresses
R> get dictionary pointer
2+ point at length
@ get length
[data-structure-field-address address length]
CMOVE move from data structure to address

FLD.MC (compile time) invoke **STR.SV**

to save characteristics of this field
(dictionary entry)
  word one = field offset
  word two = field length
(run time)
[data-structure-one-address data-structure-two-address dictionary-ptr]
>**R** save dictionary pointer
R get dictionary pointer
@ get field offset
+ add to data structure two address
[data-structure-one-address data-structure-two-field-address]
SWAP exchange addresses
R get dictionary pointer
@ get field offset
+ add to data structure one address
SWAP exchange addresses
R> get dictionary pointer
2+ advance to field length
@ get length field
[data-structure-one-field-address data-structure-two-field-address length]
CMOVE move field

FLD.MD (compile time) save field length
(dictionary entry)
  word one = field length
(run time) get field length and move between addresses

References

1. Beers, David A., "Quadruple Word Simple Arithmetic," *Forth Dimensions* IV/1.

2. Clark, Alfred, "Complex Analysis in Forth," *Forth Dimensions* III/4.

3. Harris, Kim, "Forth Extensibility," *BYTE*, Vol. 5 No. 8 (August 1980).

4. Haydon, Glen B., "Elements of a Forth Data Base Design," *Forth Dimensions* III/2.

5. Helmers, Peter H., "Userstack," *Forth Dimensions* III/1.

6. McCourt, Michael and Marisa, Richard A., "The String Stack," *Forth Dimensions* III/4.

7. McKibbin, David, "Parameter Passing to DOES>," *Forth Dimensions* III/1.

Letters *(Continued from page 5)*

A New Calendar

Dear Editor:

Jesse Wright's **CALANDER** program (*Forth Dimensions* V/4) will be useful to many people. As published, however, it is unfinished.

The method used to get the day of the week for the first day of the month is to total the number of days in each of the years from 1900 until the year being displayed, plus the number of days into that year, using **7 MOD** to give the offset. Besides being slow, it doesn't work. During September of 1989, the cumulative total of days since 1900 exceeds the magic number 32768, and thence forward **7 MOD** thinks it is seeing a negative number. The first line of dates then typically overruns the block format, producing an error indication that is difficult not to notice.

Forth is a truly marvellous programming tool, but it is no substitute for thinking out the problem. In this case, the problem boils down to calculating the number of days to advance, for a given year, into the pattern of weeks. In years not evenly divisible by four, the year begins one day of the week later; in leap years, the beginning day "leaps" ahead one more day. So, adding the number of years since 1900, the number of leap years since 1900, and the number of days into the year of interest, will give the correct value much faster and without overflow. Screen #85 includes the revised version of **DAY-OF-WEEK**. Also, the only year between 1901 and 2099 which is evenly divisible by 100 (A.D. 2000) is also evenly divisible by 400 and, therefore, is a leap year; so for the two centuries of immediate interest, the test for a leap year can be simplified to a division by four. Finally, in the interest of portability, I suggest re-naming the program **CALENDAR**

Regards,

Wendall C. Gates, PE
P.O. Box 2216
Santa Cruz, CA 95063


```
SCR # 85
  0 ( Calendar Program --- Revised  WCG  12-24-83 )
  1 : DAY-OF-YEAR   ( day, month, year -- day of year )
  2   IS-LEAP-YEAR?
  3   IF 13 + 14          ( if leap year, convert offsets )
  4    ELSE 1 ENDIF              ( else start at month 1    )
  5   OVER OVER = IF DROP DROP  ( if January, return day )
  6               ELSE DO I DAYS-IN-MONTH + LOOP ( day of year)
  7               ENDIF ;
  8 : DAY-OF-WEEK   ( day, month, year -- day of week, 0 is Sunday )
  9   DUP >R DAY-OF-YEAR   R>             ( calc days into year )
 10   1900 - DUP 3 + 4 / + +              ( calc and add leap days )
 11   1 -  7 MOD ;                ( adj. for start day, calc offset )
 12 85 .  -->
 13
 14
 15
```

Frequency Study

Dear Editor:

With what I have seen so far, the Forth-83 Standard is a definite improvement over Forth-79. I was trained in the university fashion, and would like to see Forth accommodate the "traditional" vocabulary of programming languages.

My reason for this letter is to ask if anyone has done a frequency study of Forth words. I am studying compaction techniques and would like some idea of the static frequency of occurrence of words in major packages. Typically, a few words are used very frequently and others just

(Continued on page 34)

Faster Dictionary Searches

David W. Harralson
Yorba Linda, California

The fig-FORTH model of the dictionary has a variable-length name field followed by the LFA, CFA and PFA.

(FIND) searches through the dictionary attempting to find a match to a given name. If the word it is currently examining does not match, **(FIND)** must search forward character-by-character to get to the last character in the name, and then must use the NFA to link back to the previous word. This process is very slow. **(FIND)** must look through the entire dictionary at least once for each new definition entered (twice for the fig-FORTH **-FIND**) as well as an indeterminate amount for each compiled word.

There have been proposals to provide multiple dictionary links to cut the search time. Also, Paul van der Eijk proposed in *Forth Dimensions* (III/2) changing the dictionary structure so that it looks like LFA, Name Field, CFA and PFA. I have implemented this proposal in a psuedo-fig-FORTH-83 system. Since Forth-79 does not specify the format of entries in the dictionary, this would appear to be a legal Forth-79 or -83 system.

The changes necessary, which in some cases are subtly different from those proposed by Paul, are presented here. The code for **(FIND)** is for the 8080 and is actually shorter than for the old **(FIND)**. Also, my version of **ID.** needed to be re-coded slightly. The sequence of **PFA LFA** in **WORDS (VLIST)** has been replaced with **2-** both to decrease its size and to speed it up. **TRAVERSE** is a **CODE** version that is not only shorter, but much faster. **-FIND** searches **#VOCS** vocabularies as per the **ONLY-ALSO** proposal. Another version is given for a fig-FORTH that only searches **CONTEXT** as per Forth-79, since all vocabularies chain to **FORTH**.

As far as results are concerned, loading nineteen blocks changed from forty-four seconds to thirty-four seconds, a decrease of twenty-three percent. This is not as good as Paul found, but is probably due to **-FIND** only searching **CONTEXT**.

The current words **LFA, CFA, PFA** and **NFA** are somewhat confusing, since they assume you know where you are coming from. In addition, the **2+** in **VOCABULARY** could be another word. More descriptive word definitions could be:

PFA->LFA for **LFA**

LFA->PFA for **PFA**

PFA->NFA for **NFA**

PFA->CFA for **CFA**

LFA->NFA for the **2+** in **VOCABULARY**

```
CODE (FIND)    ( here NFA --- false | CFA length true,fig-FORTH)
     D POP  BEGIN                ( GET LINK WORD                  )
     H POP                       ( HERE                           )
     D A MOV  E ORA  NO JZ       ( LINK=0 RETURN FALSE            )
     H PUSH  D PUSH              ( SAVE ADDRS                     )
     D LDAX  M XRA  3F ANI  0= IF ( LENGTHS = CONT                 )
     BEGIN  H INX  D INX         ( LOOK AT REST OF NAME           )
     D LDAX  M XRA  0= NOT UNTIL  ( UNTIL MISMATCH                )
     A ADD  0= IF                ( IF END OF STRING CONTINUE      )
     3 H LXI  D DAD              ( POINT TO PFA                   )
     D POP  XTHL                 ( GET RID OF ARGS, RETURN PFA    )
     D LDAX  A E MOV  0 D MVI    ( GET LENGTH BYTE                )
     D PUSH  YES JMP             ( RETURN LENGTH & TRUE           )
     THEN  THEN                  ( NAME NOT IT                    )
     H POP  H DCX  M D MOV       ( GET LINK AND LOAD              )
     H DCX  M E MOV  AGAIN  C;   ( NEXT NFA, SEARCH AGAIN         )

CODE TRAVERSE ( addrs direction --- new-addrs                     )
     D POP  H POP                ( DIR, ADDR                      )
     BEGIN  D DAD  M A MOV       ( GET NEXT CHAR                  )
     A ORA  0< UNTIL             ( UNTIL HI BIT ON                )
     HPUSH JMP  C;               ( RETURN NEW ADDRS               )

: NFA          ( PFA --- NFA                                      )
  3 -                          ( LAST CHAR OF NAME                )
  -1 TRAVERSE ;                ( BACK TO 1ST CHAR                 )

: LFA          ( PFA --- LFA                                      )
  NFA 2- ;                     ( BACK TO NAME & 2 MORE            )

: PFA          ( NFA --- PFA                                      )
  1 TRAVERSE                   ( FORWARD TO END NAME              )
  3 + ;                        ( POINT TO PFA                     )

: ID.          ( NFA ---                                          )
  1+ DUP 1-                    ( NFA+1, NFA                       )
  PFA CFA                      ( NFA+1, CFA                       )
  OVER -                       ( NFA+1, LENGTH OF NAME            )
  TYPE SPACE ;                 ( TYPE IT WITH A SPACE             )

: -FIND        ( --- PFA LENGTH TRUE | FALSE for ONLY-ALSO        )
  BL WORD                      ( PARSE NEXT WORD                  )
  #VOCS 0 DO                   ( SEARCH VOCAB ARRAY               )
  DROP                         ( DROP FALSE | HERE                )
  I 2* CONTEXT + @ DUP         ( NEXT VOC IN CONTEXT              )
  IF @ HERE SWAP (FIND)        ( TRY TO FIND NAME                 )
  DUP ?LEAVE THEN LOOP ;       ( IF FOUND LEAVE                   )

: FIND         ( --- PFA LENGTH TRUE | FALSE for FORTH83          )
  BL WORD  0                   ( PARSE NEXT WORD                  )
  #VOCS 0 DO                   ( SEARCH VOCAB ARRAY               )
  DROP                         ( DROP FALSE | HERE                )
  I 2* CONTEXT + @ DUP         ( NEXT VOC IN CONTEXT              )
  IF @ (FIND)                  ( TRY TO FIND NAME                 )
  DUP ?LEAVE THEN LOOP ;       ( IF FOUND LEAVE                   )
```

```
( : -FIND BL WORD CONTEXT @ @ (FIND) ; for fig-FORTH          )

: CREATE      ( ---          [FORTH83 CTOGGLE]                )
  LATEST ,                   ( LINK TO LAST WORD DEFINED      )
  -FIND                      ( PARSE WORD AND SEARCH FOR MATCH )
  IF DROP CR NFA ID. 4 MESSAGE THEN ( ALREADY DEFINED, MSG    )
  HERE DUP C@ WIDTH @ MIN 1+ ALLOT  ( ALLOT SPACE FOR NAME    )
  DUP A0 SWAP CTOGGLE        ( MAKE SMUDGED, DELIMIT NAME     )
  080 HERE 1- CTOGGLE        ( DELIMIT END OF NAME            )
  CURRENT @ !                ( LINK NFA INTO CURRENT VOCAB    )
  HERE 2+ ,                  ( CFA WORD AS IF FOR CODE WORD   )
  ;                          ( THIS IS MOD BY ;CODE OR DOES>  )

: <BUILDS     ( ---          [orphan from fig-FORTH]          )
  CREATE SMUDGE ;            ( CREATE NAME AND UNSMUDGE IT    )

: VOCABULARY  ( ---          [compiling]                      )
              ( PFA ---      [executing]                      )
  <BUILDS                    ( CREATE NEW NAME                )
  0 ,                        ( ALL VOCS END IN 0              )
                             ( CURRENT @ 2+ , for fig-FORTH   )
  HERE A081 ,                ( DUMMY BLANK NAME FOR CHAINING  )
  VOC-LINK @ ,               ( CHAIN TO PREVIOUS VOCABULARY   )
  VOC-LINK !                 ( UPDATE VOC-LINK TO NEW VOCAB   )
  DOES> CONTEXT ! ;          ( RUN TIME, UPDATE CONTEXT(0)    )

: WORDS       ( ---          [fig-FORTH VLIST]                )
  CR CONTEXT @ @ BEGIN       ( WORDS IN VOC(0)                )
  C/L OUT @ -                ( SPACE LEFT ON LINE             )
  OVER C@ 1F AND 7 + <       ( WILL NEW WORD FIT ON LINE?     )
  IF CR THEN                 ( NO, ISSUE CR - SETS OUT=0      )
  DUP DUP PFA 4 U.R SPACE    ( PRINT PFA ADDR                 )
  ID.                        ( AND NAME                       )
  OUT @ NOT OF AND SPACES    ( PAD TO MOD16 BOUNDARY WITHOUT / )
                             ( EXCEPT 1ST WHICH IS MOD15      )
                             ( HELPS IF LAST WORD BARELY FITS )
  2- @ DUP 0=                ( MORE WORDS IN VOC?             )
  ?TERMINAL OR               ( OPERATOR HIT BREAK?            )
  UNTIL DROP ;


CODE (FIND)   ( here NFA --- here false | CFA -1|1, FORTH83   )
     D POP  BEGIN              ( GET LINK WORD                 )
     H POP  H PUSH             ( HERE                          )
     D A MOV  E ORA  NO JZ     ( LINK=0 RETURN FALSE, HERE     )
     D PUSH  D LDAX            ( SAVE ADDRS                    )
     M XRA  3F ANI  0= IF      ( LENGTHS = CONT                )
     BEGIN  H INX  D INX       ( LOOK AT REST OF NAME          )
     D LDAX  M XRA  0= NOT UNTIL  ( UNTIL MISMATCH             )
     A ADD  0= IF              ( IF END OF STRING CONTINUE     )
     H INX  XCHG               ( POINT TO CFA                  )
     D POP  XTHL               ( GET RID OF ARGS, RETURN CFA   )
     D LDAX  40 ANI            ( GET LENGTH BYTE, GET PREC BIT)
     YES JZ                    ( RETURN -1 IF NOT IMMEDIATE    )
     1 H LXI  HPUSH JMP        ( RETURN 1 IF IMMEDIATE         )
     THEN  THEN                ( NAME NOT IT                   )
     H POP  H DCX  M D MOV     ( GET LINK AND LOAD             )
     H DCX  M E MOV  AGAIN  C; ( NEXT NFA, SEARCH AGAIN        )
```

End Listing

Revisited:

Recursive Decompiler

Norman L. Hills
Des Moines, Iowa

The decompiler is one of the most useful tools in the Forth toolbox. The recursive version presented here is a combination of ideas from three previously published decompilers, with a few additions. Most of the hard part was done by Ray Duncan (*Dr. Dobb's Journal*, September 1981), Robert Dudley Ackerman (*Forth Dimensions* IV/2) and the SYN-1 Users Group (*Forth Dimensions* III/2).

To use the decompiler, type **RDCP cccc** where **cccc** is the word name to be decompiled. If the only screen response is **cccc**, the word is a code primitive. If the word is a variable, constant or user variable, this fact is displayed with its current value shown in hex. "Not in dictionary" is a message with obvious meaning. Otherwise, the decompilation begins with the display of : and the address of the **CFA**. All values and addresses are shown in hex. If the word being decompiled is an immediate word, this fact is shown with the :. Values are displayed for referenced variables, *etc.* as well as for branch addresses and literals. If a referenced word is an immediate, **[COMPILE]** is shown before the word, since this would be required to compile the word.

Control of further decompilation is the result of responses to **KEY**. Q (for quit) ends the decompilation, regardless of the nesting level. A carriage return produces the "recursive" part of the process by starting to decompile the last word displayed. As the reverse of this, U (for up) ends decompilation of the current level and continues processing the prior level. Any other reply displays the next word in the current level.

The code is in Forth-79, except that it assumes a fig-FORTH dictionary structure and **NFA**, **PFA** and **CFA**. The **CASE** structure is that of Dr. Charles

(Continued on page 18)

```
RDCP   RDCP

3373   :
3375   BASE user 10
3377   @
3379   BASESAV var A
337B   !
337D   HEX
337F   -FIND
3381   OBRANCH   33A1
3385   DROP
3387   0
3389   GIN var 2
338B   !
338D   CK:
338F   OBRANCH   3399
3393   (GOESINTO)      ( CR entered )
329B     :
329D     CK:           ( CR entered )
31E3       :
31E5       DUP
31E7       CFA
31E9       @
31EB       LIT   A24
31EF       CFA                ( U entered )
329F     OBRANCH   3368
32A3     2-
32A5     DIN           ( CR entered )
3085       :
3087       CR
3089       DUP
308B       .G@
308D       SPACES
           ;S
32A7     (.")     :
32AD     2+D           ( U entered )
3395   BRANCH   339D
3399   NFA         ( Q entered )
```

Sample Run of RDCP
Space bar entered unless otherwise noted

(See Listings on page 17 & 18)

```
Screen # 15
 0 ( Recursive Decompiler - 1 )      HEX
 1
 2  VARIABLE  GIN    ( No. to Indent )
 3  VARIABLE  BASESAV    ( Save Original BASE )
 4  VARIABLE  KEYSW    ( Continuous or Step Switch )
 5  VARIABLE  RDFEN    ( Recursion Fence )
 6  ' C/L  2- @ CONSTANT  CONST.ADR  ( Run-time CFA )
 7  ' BASE 2- @ CONSTANT  USERV.ADR  (        "       )
 8  ' GIN  2- @ CONSTANT  VAR.ADR    (        "       )
 9 : U.   0  D.   ;
10 : GOV   OVER  @  2+  ;
11 : 2+D   2+  DUP  ;
12 : .G@   0  4  D.R  GIN  @  ;
13 : DIN   CR  DUP  .G@  SPACES  ;
14 : GIN+  CR  OVER  .G@  2+D  GIN  !  SPACES  ;
15 : MYSELF   LATEST  PFA  CFA  ,  ; IMMEDIATE   -->

Screen # 16
 0 ( Recursive Decompiler - 2 )
 1 : GCHKTYP    ( pfa --- f )  ( Check for following literal )
 2     CASE    ' LIT       OF  1  ENDOF
 3             ' BRANCH    OF  1  ENDOF
 4             ' OBRANCH   OF  1  ENDOF
 5             ' (LOOP)    OF  1  ENDOF
 6             ' (+LOOP)   OF  1  ENDOF
 7             0  SWAP ( ff return - ENDCASE drops addr )
 8     ENDCASE  ;
 9
10 : 1KEY    ( pfa --- pfa c )   KEYSW  @
11       IF  ?TERMINAL  IF  ASCII Q
12         ELSE  DUP  RDFEN  @  U<
13         IF  BL  ELSE  0D  THEN  THEN
14       ELSE  KEY
15       THEN   ;   -->

Screen # 17
 0 ( Recursive Decompiler - 3 )
 1
 2 : GCHK    ( pfa --- pfa [altered] )   1  ( true for case )
 3     CASE    GOV  ' COMPILE  =
 4       OF  2+D  @  2+  NFA  ID.  ENDOF
 5           GOV  GCHKTYP
 6       OF  2+D  @  SPACE  U.  ENDOF
 7          ( GOV  ' CLIT  = )
 8     ( OF  2+D  C@  SPACE  .  1-  ENDOF )
 9           GOV  ' (.")  =
10       OF  2+D  COUNT  TYPE  DUP  C@  1-  +  ENDOF
11        ( endcase will drop unmatched tf )
12     ENDCASE   2+  ( increment pfa )
13     -2  GIN  +!   ;
14                              ( pfa --- pfa  f )
15 : CK:   DUP  CFA  @  ' :  CFA  @  =   ;   -->

Screen # 18
 0 ( Recursive Decompiler - 4 )
 1
 2 : DISTYPE    ( nfa --- )
 3          DUP  C@  40  AND    ( get 'immediate' bit )
 4     IF  ." [COMPILE] "  ID.
 5     ELSE  DUP  ID.  PFA  CFA  DUP  @
 6       CASE  CONST.ADR  OF ." const " EXECUTE  U.  ENDOF
 7             VAR.ADR    OF ." var " EXECUTE @  U.  ENDOF
 8             USERV.ADR  OF ." user " EXECUTE @  U.  ENDOF
 9             DROP
10       ENDCASE
11     THEN   ;      -->
12
13
14
15
```

Eaker (*Forth Dimensions* II/3) and the following definition

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

(insert **HERE** after **WORD** in fig-FORTH) is from Raymond Weisling (*Forth Dimensions* III/3). The parentheses in **GCHK** should be removed if a byte literal is available in your system.

Storing a non-zero value in variable **KEYSW** produces a continuous decompilation without keyboard intervention. This is probably most useful when **EMIT** is directed to a printer. A second variable, **RDFEN** is used in continuous mode as a fence to prevent looping in the innards of Forth. This variable may be changed *with care* to show more (or less) detail.

RENEW TODAY!

Listing *(Continued from page 17)*

```
  Screen # 19
 0 ( Recursive Decompiler - 5 )
 1 : (GOESINTO)      ( pfa --- )     CK:
 2   IF  2- DIN ."    :"  2+D  NFA  C@  40  AND
 3    IF  ." IMMEDIATE"  THEN
 4   BEGIN  DUP  @  DUP  ' ;S  CFA  =
 5          OVER  ' (;CODE)  CFA  =   OR  0=
 6   WHILE  ( High Level & not end of colon definition )
 7      2+  GIN+  DUP  NFA  DISTYPE
 8      1KEY     CASE
 9        ASCII Q OF  SP! BASESAV @ BASE ! QUIT  ENDOF  ( Get Out )
10              0D OF  MYSELF  ENDOF  ( CR -  Go down a level )
11        ASCII U OF DROP DROP R> DROP -2 GIN +! ENDOF ( Go )
12        DROP  ENDCASE   GCHK                  ( up a level )
13   REPEAT  CR
14     GIN @  6 + SPACES  2+  NFA  ID.
15   THEN   DROP   ;         -->

  Screen # 20
 0 ( Recursive Decompiler - 6 )
 1
 2 : RDCP   BASE @  BASESAV !   HEX
 3          -FIND
 4    IF  DROP  0 GIN !  CK:
 5      IF  (GOESINTO)
 6      ELSE  NFA  DISTYPE
 7      THEN
 8    ELSE  ." Not in Dictionary"
 9    THEN
10    CR  BASESAV @  BASE !   ;
11
12    HERE  RDFEN  !  DECIMAL  ;S
13
14
15
```

Interview

William F. Ragsdale

William Ragsdale has served as President of the Forth Interest Group since its inception. Curious about a man who has spent several years donating a good portion of his time to further FIG's growth, in February Forth Dimensions *cornered Bill in his office at Dorado Systems in Hayward, California.*

I've heard how the Forth Interest Group began, but never why ...

The "why"? There isn't any why. It just happened, it just grew. The reasons weren't compelling, other than a recognition of the common need for the communication of information by people. The "how" was that I was doing a lecture at a computer club on high-level languages and structured programming in about 1977 or '78. I used Forth as my illustration. At that time I had just gotten a very minimal level of Forth running at work and was kind of proud of that. I brought it up and showed it running. I loaded a paper tape, Forth said "ok" and ran very, very minimally.

John James and Dave Boulton were present at that meeting. John had not heard of Forth before but he was interested in it philosophically. It just had the right aspects that he thought were appealing. Dave was using it professionally. He was with a company that was using Forth on the San Francisco peninsula. So the three of us had a chat and said, "Let's get together some other time and talk about it." Well, someone there suggested that I give a short talk at the Homebrew Computer Club. After that talk, the people who were interested got together and said, "Let's have coffee at the Stanford student lounge." So we went and sat under an oak tree. There was Kim Harris, Dave Kilbridge, John James and Dave Boulton. They were the founders of FIG at that time. A couple other people were present, too; one fellow named Tom Olsen had a running Forth system on a PDP-11 at home in his living room, really loaded with three floppies, hard disk drive, CRT, DEC Writer, just loaded.

Well, we said, "Let's get together to talk about it some more and to see what our common interests are." That was about a month later. I remember it was Super Bowl Sunday, 1978, when FIG was founded in an apartment in Sunnyvale. At that point we vowed to write a newsletter. John James wrote the first couple of articles, and five or six of us developed an editorial that kicked off *Forth Dimensions.* Tom text processed the first couple of issues on his PDP-11. At that point, my wife Anne and I took over the printing, publication and mailing of *Forth Dimensions.* We got the material together and did the next several issues. Sadly, along the way Tom died of diabetes. When we lost him, we lost one of the team members there at the start. So the five of us carried on.

The West Coast Computer Faire came along about three months later. At that time we had a couple of issues of *Forth Dimensions.* We were giving it away then — the first three issues were free because we didn't know the longevity of it. We got about two hundred people from the computer fair to join or subscribe. Then the following year, we really started swinging.

How did you put out information about Forth in the very beginning?

We announced one Saturday, "Let's have a class on Forth, ten o'clock on Saturday morning at Dorado Systems." About thirty people showed up. We had a room full of people. We had every chair, all the sofas, all the floor space filled. So we gave a talk on Forth. I had a demonstration system available, so they could see it running in a minimal way. We had a couple of hours of teaching, but then we polled the people and asked, "What access do people have to Forth? How many people have a running Forth system?" Out of thirty, I think only three raised their hands: me, Dave Boulton and Tom. I was dismayed, because we knew that from a learning standpoint, people would leave the classroom, go home and a week later they would have trouble remembering all they had learned.

A short time thereafter — it might have been at that meeting — the first microcomputer implementation of Forth was made available for the Pet computer. *Forth Dimensions* immediately started getting letters telling us how terrible the product was and how people hated it. That put us in the worried mode. Here was a vendor proclaiming loud and clear that he was selling a Forth system, which was very poor, for personal computer use. We had contact, by that time, with Elizabeth Rather from FORTH, Inc. We queried their future role, would they have Forth for the Apple and the Pet, would they support personal computers? She very clearly stated, "absolutely not," that was not their mission and they would not support personal computer versions. They sold systems costing $2000 - $10,000 to scientific and industrial areas.

History has borne that out, although they did come out almost five years later with a version for the IBM-PC. But in the interim they have not supported any other personal computers with a commercial product. These two factors — first, the potential for people who don't know any better to put out terrible implementations and, secondly, the leader in our industry saying that they were not going to be active in personal computers — meant that to learn and to teach Forth we were absolutely up a creek, because there were going to be no commercial sources of the language.

So FIG decided to do its own implementations?

Not quite. We wanted to supply guidelines to suitable vendors, but none existed. I decided to "salt the gold mine." By that time, I had gone through three releases of our system here at Dorado. It was up to version

3.6 and by that time it was turning into a usable system. So after careful consideration, my opinion at that time was to release Dorado Systems Forth version 3.6 into the public domain, and that became the foundation for the FIG model. In fact, the other day I had reason to go back and refer to it and, if you look at about block #54 of the FIG model, the abort message says, "Forth 65 version 4.0." Forth 65 was our in-house name for it, version 4.0 was a cleaned-up 3.6, and that became fig-FORTH 1.0.

With the release of that into the public domain, we needed multiple, portable implementations. Ours was written in 6502 and I was not competent to write a version for six or eight other processors, technically or from a time standpoint. So in *Forth Dimensions* we put out a call for participation that announced the project and said, if you have assembly language background and high-level language interest — just the interest — then come to this workshop.

We got a lot of respondents and screened them down to two people per processor, a primary candidate and a backup. We had the 8080, 6800, PDP-11, Pace, 1802, TI 9900, with two people covering each — like Noah and the ark, it was two by two. We then started what has turned into the Saturday FIG meetings in Northern California. Kim Harris was our librarian and took meticulous notes as the work expanded and as the information was developed and fine tuned. We used that document to go back and analyze and correct our notes. That provided an invaluable record of the sessions.

We walked people through one section at a time, and when they ran into problems we swapped answers and analyses. We looked at machine-portable solutions. Out of that, two improvements were made before the final release. Dave Kilbridge worked on the fixed-point arithmetic, because in the original implementation some of the signing, carries and so on needed improvement. By that time we had contact with the people at the State University at Utrecht in the Netherlands. They swapped with us the entire source code for their system, some 2000 screens of it, with permission to utilize portions in our own implementation. We took the compiler security from their code; until that time, none of the USA versions had that.

The first running Forth system thirteen of the team members ever saw was the one they had developed. We had a few ground rules in there that everyone adhered to meticulously. It worked out very nicely. They would basically follow the model faithfully and minimize any deviations from that model because we realized that it was to be a Rosetta stone, where we would have the same information presented in a variety of forms. We even got to the point of having the very same assembly language labels throughout, and one of our very late sessions was to agree on the labelling so that all read the same; they were to be in the same order and so on.

So the University at Utrecht had Forth that early on?

Yes, it grew from Chuck's original work at Kitt Peak. Astronomers from around the world would spend time at Kitt Peak. Depending on what machine they had at home, they would take a tape back with them around the world and, bingo, they had Forth. Then Forth was up at St. Andrews in Scotland, at a couple of places in France, in Chile and CalTech. This was all before the FORTH, Inc. days, from 1969 to about 1973. Forth started its migration through the world of astronomy.

Forth then was really an operating system for what I call a crippled computer. Forth has been treated in a receptive way by users of computers with very limited resources in terms of memory, mass storage or input/output. The Varian 620i was a crippled computer. Some of the early Hewlett-Packards, the 2100 series for instance, had very limited manufacturer support. In such cases, Forth has been graded with very high marks.

On the other hand, we in the Forth community face a very real problem, in that as the manufacturers have provided increased quality of software, the need and demand for Forth appears to diminish. Forth was providing some irreplaceable attributes five years ago. Now it appears that a number of those attributes are no longer as attractive as they were. For example, there is more memory space available, I/O is faster, more disk space is available, file structures are less limiting. This puts an increased challenge on people using and writing Forth systems. Are they going to stay back in the "crippled computer" mentality or are they going to continue to grow and follow the industry needs?

Considering the attributes of Forth and of today's computers, is the language still contemporary?

Forth has the proper elements, but I fear people are not making anywhere near the use of them that they should be. We are coming around a five-year circle, back to where in 1979 Hans Niewenhuijzen said we would be. Forth itself was a foundation, a base on which higher-level language constructs would be developed. Hans was espousing that idea at meetings and the message didn't sink in very well. Hans was saying, "Look at these other languages, Pascal, Algol, Lisp; see the components of them that are neat, pull them into Forth and then build Forth into a higher-level language." Around here, the message didn't really take root very well and Hans was disappointed as a result of it. He felt that people didn't appreciate his ideas. I think it was really more that they didn't see and appreciate the deficiencies of Forth.

But the last five years have shown me that the deficiencies aren't in the language itself. It's in the utility aspects of the language. That is, the absence of a file system, the limited choices that you have to manipulate amounts of information larger than just a number on the stack or a character string. Some of the other programming environments, such as UNIX, allow you to manipulate files, or to pipe or convey information, and to modify it from one file structure to another on its way from input to output. These much higher-level programming environment constructs have not been carried forward in Forth. So Forth has stagnated over the last four or five years. The improvements in Forth have been very modest compared to the user utility we find in other programming environments.

The worry to me is that we are slowing down, and that users now cannot

do much more with Forth than they did four or five years ago. The major question is, how do we create the attitudes, the interests and the environment for people to build on a Forth system, to make it more and more interactive, to be more and more useful and to manipulate higher-level constructs than just numbers and a few characters?

Are you saying that some of our efforts toward standardization should be re-directed to encourage more creative approaches?

Not at all. No, I think the Forth Standards Team's effort, as it is presently going on, is exactly the right approach at exactly the right level. It is not too low level, just at the code and a few primitive words; on the other hand, it is not into very high-level constructs such as trying to have a standard editor or trying to have standard telecommunications protocols. So I think it is just right. There is enough in the standard to write useful applications that are fundamentally standard even though small parts of them may have to be specialized enough that they are non-standard.

We are now ready for a new effort level and a new mutual agreement between users and vendors on such things as a file structure, and an allocator mechanism for mass storage. That is, files with limited sets of rules associated with them. Not "no rules," but with a modest and simple rule set so that I can generate variable-length data structures called files. On the other hand, I should still be able to hold data in blocks if I wish. This means that I would ask the system to generate a file for me and I can put data into that; or I can ask the system to give me some blocks because for some applications the blocks may be better. I can ask for contiguous blocks or scattered blocks, and when I am done using them I can return them to the system and say, thank you very much. That's a minimal next step.

Beyond that, we start to look at the user interface to a computer and how it can be more responsive to our needs for such things as graphics, sound, pointing devices such as a light pen, mouse or graphics tablet. How are these going to be addressed or utilized in a Forth environment so that we are not constrained by one specific set of hardware?

Will these things come to pass, or will they require a fifth-generation language?

No, no, no. No new breakthroughs! Everything in manageable, bite-sized, modular pieces. No new fifth language, no throwing the baby out with the bath water. But I think it is very appropriate for us all to question what is the right environment, and to encourage things like this to happen. Will it come through the present vendor channels, will it come through academic support, will it come through users groups, will it just come through innovative individuals working on their own? The avenue through which these things are likely to come is not at all obvious to me.

There is a tremendous range of opportunity for the vendors to build in this area. For example, an interesting one people should look for is Pierre Moreton, who is currently working in Palo Alto, California. He is publishing a portable Forth file system that he has developed over the years. People who utilize it say it is a very good work. I'm not advocating Pierre's file structure as the standard Forth system, my point is that people immediately see and value his work, appreciate it and begin to use it. That demonstrates its merit. Examples of that sort of an enhancement are fairly limited; that is, there have been very few of them so far and there is a tremendous opportunity for more. Ren Curry's programming tools are spoken of very highly by users who would not have a running system now without that sort of a debugger, breakpointer and decompiler.

The vendors now walk a fine line between what is good Forth and what are good contributions for products of the future. I think they are playing it too conservatively. Right now the vendors are just trying to do a good job on Forth and to document it; they are pulling back too soon. They are not meeting the challenge, not addressing unfulfilled needs. I do not know where there is a commercially available spreadsheet in Forth. There is QTF, the text processor from Leo Brodie; on the other hand, that works on blocks and is a fairly constrained text processor. It is enough to get the job done, but you wouldn't use it if you had better alternatives. I think there is a tremendous opportunity for someone to come along with a good Forth system as a basis and to add the interfaces users expect.

What is it that enables you to go into a situation like meetings of FIG volunteers, where there may be an emotional charge in the air, yet come out of it with a concensus and everyone still working together? There may be arguments and debates, but votes are taken, a decision is reached and people continue to work together.

It may be a little corny, but the motivation and common interest we have in Forth really transcends a lot of these short-term individual priorities. I would not at all allude to the fact that it is a management style, or an environment that is created which makes this happen. It falls strictly back to the spirit of the individuals; there is a bit of a filtering process that goes on, because a number of people who may have significantly contrary points of view just don't continue to participate. Therefore, we have people who think in a fairly consistent fashion, and who are able at least to see the other fellow's point of view.

The volunteer aspect, you see, is a major element, in that people realize that they have to work for their own self interest to enjoy participating. We have to recognize that and allow it to play a role. They are doing it because they want to and because it is needed, not because it is required by someone else. They also do it because there is a feeling of dependence: for whatever function they are performing, other people are depending on them to do that. For example, the team that puts the FORML conference together has done so year after year because they see a very great need for that sort of forum to exist, and for the proceedings to get published and that a diversity of opinions be represented at the meeting. It is common desire and common good that makes it flourish.

I see us at a major turning point in the interest group's development. Until now, of necessity and correctly, it has been fairly focused in Northern California because we had enough people

here to reach critical mass to get the work done. Now we are at a point where we can start to distribute the participation totally through the membership and to strengthen FIG as a membership-driven organization — one that is controlled and operated by, and dependent on, its membership. Up till now it has been organized so that it is the central FIG planning group that provides services to members: publications, listings, conference proceedings and major events. I think the pendulum is now swinging so that work such as the chapter development done by John Hall will begin to bear fruit. I hope that in the next few years we will have our national convention regionally so that it will be in Washington, Dallas, New Orleans. We will know we have an effective membership organization when we can move around the country and have the organization effort done regionally.

Why do you expend such volunteer effort on behalf of FIG? Some people are surprised at your investment of seven years.

I've heard that more than once. Some suspicious types feel there must be something sinister in a sustained effort for that period with no financial compensation. The obvious answer, at least to me, is that I enjoy communicating, both in written and spoken form. In order to talk about Forth, there must be others, so a growing group is necessary for all of us to have a forum.

How do you view the role FIG now serves?

There is still a great deal of understanding being sought about FIG's service role and how it is changing. Early on, it was to provide a newsletter and education to people, to fill that need of how do people learn about Forth, how to use Forth in a productive way to meet their own needs.

One question currently being debated with a great amount of interest is whether the interest group should engage a significant amount of its resources in the popularization of Forth to people who are otherwise unacquainted with it. Or should its role be devoted primarily to serving members' needs, people who are already familiar with Forth and who wish to enhance their own understanding and skills? That is, are we on a public relations-oriented mission to legitimize Forth and make people aware of it who otherwise would have no contact? Some say that is a more appropriate role for the vendors; that the interest group comes into play once someone has an awareness of Forth and desires to enhance his skills. There is certainly no answer to those topics yet, and the more points of view we receive, the better off we will be.

Why hasn't FIG outlived its usefulness?

There is an easy answer to that. It's the only game in town. It is the only community of interest that has developed associated with Forth. The standards team is a group of only about twenty-six people with periods of intense activity and periods of latency in between. One manufacturer has started and supported a users group. I am not familiar with the exact structure of it but the activity, from all indications I see, is very modest. But FIG, as I said, is the only game in town. Does that sound piggy?

The membership aspects of FIG are just being tapped and developed. I feel very firmly that over the next two to three years we will see a major change in the complexion of FIG in its interaction with and its dependency on its membership. We have a number of elements going into effect this year, such as membership cards, discounts, possibly expanded hotline services based on membership. There are half a dozen or so aspects that we are trying to strengthen for the membership. It is a case of two plus two equals five — if the people gather together, they get more results from their common interests.

Is FIG likely to endorse Laxen and Perry's F83 implementation as a language model?

I certainly hope we will. The standards team, I think, missed a boat: upon the team members' acceptance of the draft, they felt their job was done. It is becoming apparent to me now that it wasn't. There is another stage left, which is to encourage ratification, if you will, of the standard draft by communities of users. The standard really isn't a standard until it is accepted broadly and used broadly. One of the goals of the Forth Interest Group is to take a formal action on behalf of our membership with regard to the standard, to set up a ratification process to examine it against our mutual needs. Then if this review team considered it appropriate, it would come out with an acceptance statement on behalf of the membership and the interest group, endorsing the standard and recommending its use.

What is your relationship with other organizations, for example Forth Inc. and the Forth Vendors Group?

I purchased stock in Forth Inc. in 1979 and was subsequently invited to join their Board of Directors. I served as Director from 1980 through late 1982, and then as Chairman until October of 1983. My only current involvement is as an investor.

I joined the Forth Vendors Group at its inception, about a year ago. My product is the Ultra Compiler, a target compiler, which is made available to industrial customers. One vendor currently re-sells it as part of a complete system. I'm sure I'll have further involvement with Forth applications.

Does a marketplace exist for Forth and Forth products?

The market is developed for Forth programmers, but not for Forth products. Right now, there is a sufficient number of industrial uses for Forth that professionally skilled Forth programmers don't seem to have any trouble getting a job. I know of two industrial companies which were considering dropping Forth as their programming language in the past year because of their difficulty recruiting people. The projects had been started in Forth by someone who was skilled in it, the person left the organization and the firm had such difficulty in supporting the activity with other people, that they were very insistent about dropping Forth.

The skills for Forth programmers are merchandisable. On the other hand, it is not at all obvious to me that there is truly a marketplace for Forth itself, but I have a lesser reservation on Forth applications. I would substantiate this by looking at the sales of companies such as Digital Research and

Microsoft. They are providing languages and operating systems, essentially productivity tools for other programmers. Well, that's what Forth does. Now, if we take the sales of the top ten companies in the Forth community, filling in your own names as to which they may be, we would add up their total gross sales at maybe something like three million dollars max, and possibly something closer to two million dollars. Digital Research and Microsoft are 6000 percent larger than us. So another way of saying this is that the Forth community has to grow by, say, a hundredfold; there is a hundredfold growth opportunity for the Forth community to come into its own in the market.

Now if we are saying that Forth is so dynamite, so productive, so innovative, so useful, solves so many problems, why do we have this hundred-to-one gap between Forth and the other environments? The challenge is essentially on the vendors' shoulders. They have to make products that are responsive to real-world needs, not their own needs, not their staff's needs. They have to define their business activity by the nature of the customer marketplace they want to serve. Right now it is obvious that they are defining the nature of their companies in terms of a very small number of customers with a very limited need. To me it is a pity, because the Forth manufacturers choose voluntarily, in the way they conduct their businesses, to define their service role in such a narrow way.

Perhaps some of the innovation you mentioned earlier should come about because people listen to public needs...

That's good. As you said those very words, when you came to the "innovate" part I winced a little bit, and then you got to the part about "customer needs" and I smiled a little bit because that is right. We have, I think, been polishing a precious gemstone and telling everyone how pretty it is, rather than being responsive to the type of jewelry people really wish to wear. Now, I'm not at all advocating that we throw over the precepts of simplicity and portability and generality that make Forth so strong. But we have a tremendous resource to be used for solutions that we have missed so far to users' problems and productivity needs. A few vendors have responded in the past to these sorts of customer needs, but it looks to me as if the growth curve that the microcomputer industry is currently enjoying, something like thirty-five or fifty percent per year, is not going on within the Forth vendor community.

Can someone going into business as a Forth vendor look forward to a substantial profit? Or is that premature thinking?

I see an opening as big as a barn, a gigantic pot of gold at the end of the rainbow, or Ali Baba's treasure cave. It is the tremendous financial reward that can be achieved by the use of Forth that, unfortunately, isn't being realized. They seem to be solving three-year-old problems with two-year-old solutions. They are not looking where the action is or where the needs are.

I'll point out elements that I feel are components of this. I see a system with an integrated file structure and data base engineered by ordinary good practice, contemporary practice that you can get by reading texts coming out in the university community. Components are two-phase record locking and transactions with roll-back and roll-forward journalling. That is then coupled with a compatible query language and virtual execution. Not the Forth style of access to disk blocks and data storage, but the ability to execute code out of virtual memory. A good combination of these elements would be competitive in the operating system arena equivalent to, say, the UNIX system.

I see vendors such as Unisoft and Relational Technology in Berkeley that are currently doing four to five million dollars a year, and are heading toward doing fifteen to twenty million dollars a year in what is very high margin work. Once you write the code, you then have multiple uses of it. You are licensing a couple of hundred other companies to go out and sell your system. These two companies are going from startup into two and twenty million dollars a year in sales and I see no reason whatsoever why this cannot be done with a Forth starting point. So, with those grandiose statements, how do we inspire the vendors to set their goals high enough, to set their desires to a sufficient level that they will be able to go out and achieve that?

It's a tall order. The technology components are available. So it looks like it's going to need an accident. Maybe we just need some luck, where the right innovator drops into the right business environment coupled with the right sensitivity toward customer needs, and it takes off.

Yet many of the people who use Forth have chosen to avoid large systems...

I'm not talking about big environments. The Pick operating system has many of the characteristics I just mentioned and is smaller than Forth. It runs with a 4K nucleus and 250K of code. It has virtual execution, data base, query language, text processor, multi-tasking. When someone wants to use that system, it is a million dollar license. Now that's leadership. They have customers lined up. They used to do one implementation a year, now they do four a year and they have them lined up and waiting. If Dick Pick and Chuck Moore had gotten together on day one, a miracle would have resulted.

Why isn't this being done in Forth? Well, I'm not sure. It may be that innovations occur when people are in uncomfortable situations, when they are boxed in. Then they have to find solutions, whether they are technical, managerial, people or financial. Maybe we just have to go along until the right combination of people are in that box.

If I'm so smart, why don't I go do it? Well, it's not easy, it's a Herculean task — it's building an industry. Up till now, Forth and Chuck Moore have built an avocation. How are we going to make an industry out of it? That's why I did the paper at FORML on leadership, trying to get that inspiration going.

Forth right now is like a well-worn, comfortable slipper. It is convenient, easy, fun, all that. A new order must form to respond to needs. What we need is another Ice Age, some more freezing people struggling to kill another mastodon for dinner.

What do you personally want from FIG?

It has gone on too long with me calling the shots. That doesn't help FIG, it doesn't help me. So we have got to find a way to ease me out, and the best way is by having people who grow and develop, and who want to participate and pick up the interest, skills and rewards from doing it on a public-service basis. I think that's what FIG needs now. It's like a relay race and we now need a continuing team. Seeing John Hall come along is a breath of fresh air. He just came in at the right time and the right place, picked up the challenge and responded to it and is doing very well. Two or three years ago, Kim Harris did that with FORML. Right now, we need a little horsepower to aid the plain old business management. For a while I did it, then we turned it over to Roy Martens and now it looks like we are in the next generation of that area.

Beyond that, I fall back on platitudes. We want a membership-based organization that has sufficient international strength and depth that we provide mutual support to one another. That is, we have the right people skills, the right technical skills, the right communication skills so that *Forth Dimensions* flourishes, the chapters interact at a people level and at a chapter level. Then we have one or two international events a year so we interact on a global level. It is just like Forth words: you have the code words and the high-level words, the user interaction at the topmost level, everything layered up. And it's the same way with the group activities on local, national and international levels. The Forth Interest Group has hit a level of support and need such that it will continue unabated, it will flourish of its own momentum.

Are there any limits to growth? I'm not sure on that. There must be, the day will not come that there are a million members of FIG. I use the book *Starting Forth* to test our potential universe. I understand there are fifty or sixty thousand copies in print, and everybody who owns a copy ought to be a member of FIG also. When we've done that, maybe we've reached our limit. With a group that large, the benefits go up exponentially.

What kind of people are attracted to Forth, and which do not do well with it?

I see two types of people that seem to find Forth valuable. The first is the innovative or ingenious person, a sort of puzzle solver who takes as much relish in the method of solution as he does in the solution itself. I can write a program in BASIC, in Fortran and in Forth, and yet I'm proudest of the Forth solution when I am done. That is the one I would show to somebody. The other type of person is one who appreciates the flexibility of Forth. In some cases they are a little more rebellious or non-conformist, they don't want to solve the problem the way the other person does it. They want the liberties Forth provides to develop their own style of solution, their own method of expression. Rather than showing off the solution, they tend to be most content with the fact that they

have solved the problem in their way. The latter type also is more heretical or non-conformist in that they are prouder of the solution if it is a little out of the mainstream.

What are your biases as a Forth programmer?

I have the same bias that most have, and it's not a good one. That is the attitude that I can do it better, that my code is, by definition, better than your code. Now I have enough perspective to joke about it and to realize it. But people re-invent the wheel and then feel that the best solution to any problem is the one they just thought of. Forth itself encourages people like that. It comes with the territory, and it gains us the insights of bright, creative people. On the other hand, it means that we lose the ability to build and utilize the work of others. I see a tendency for people to either implement their own system or install their own system, rather than to search for and utilize the benefits of vendors' systems. If more people made better and more extensive use of vendors' products, it would improve the quality of the product they get and there would be a stronger and better marketplace.

One illustration of that: I flip open a magazine such as *Dr. Dobb's Journal* and I see in there six implementations of C, one of which sells for $350, and the rest for $500 or $600. Then I take the same magazine and look at Forth implementations. I see about six implementations of Forth, the lowest cost for which is $35 and the most expensive, I believe, about $195, with most about $85 or $100. Well, pricing often can be a clue to perceived worth. What this tells us is that the purchaser perceives that in a Forth system he has only one-fifth the value he would have in a C system. That's not a good testimony. We want to demonstrate the value of the Forth systems in such a way that the perceived value goes up and Forth systems are now worth every bit as much as C systems.

Biases part two: "I can do it better" is typical of the individualist aspect in the interest group. We recently had a planning session at which these topics were discussed, and it was brought out at the meeting that one of the characteristics of the Forth programmer is that he would prefer to modify his programming environment to match his own needs rather than moderating his needs to match those of the environment around him. I don't think this is necessarily a negative characteristic, but I think it is one that we should carry in the back of our mind. Where could we use that as a strength to build on?

There's a saying that where programming is concerned, better isn't always best. Refinements of the design and code can go on and on, with the project never reaching completion.

A classic quote goes like this: Hardware designers and engineers are used to standing on each other's shoulders, building from components supplied by other people. Programmers tend to stand on each other's toes. They get in each other's way and each individual makes only a small contribution. How do we get to the point where we are standing on each other's shoulders and each one is getting a two-to-one multiplier over the productivity of another?

The people in the Forth Interest Group have changed, and the collective nature of the group has changed. In the 1979–1980 era, we were very heavily in an evangelical or missionary role. We were at meetings saying, "Look at this, read my code, Forth is great!" It was out of pride, and it also reflected the fact that many people had not had any exposure at all to Forth. I'm pleased that changes have occurred, and that now we are less strident in our tone, less vocal. We are now providing information to a more receptive audience who is saying, "Say, I've heard about Forth. Can you help me to understand more or to get some use from it?" That is a remarkable improvement in the short space of three years.

SEND A CHECK TO FIG TODAY!
MAKE THIS YOUR BEGINNING!
RENEW NOW!

DO...WHEN...LOOP Construct

R.W. Gray
Los Angeles, California

This article will introduce a new Forth loop construct which is clear and compact. This construct can have the forms shown in figure one.

These constructs are a result of an attempt to implement the Forth-83 **LEAVE** which exits the loop immediately; plus the realization that every **LEAVE** must be contained in an **IF ... THEN** type of construct. Thus, the screens contain definitions which implement a conditional Forth-83 Standard **LEAVE**. The **DO ... WHEN ... LOOP** construct can be seen as similar to a **BEGIN ... WHILE ... REPEAT** structure.

A problem with the Forth-83 **LEAVE** is that it requires improper nesting of conditional statements. Since **LEAVE** must be in a conditional statement like:

DO ... IF LEAVE THEN ... LOOP

and the **LEAVE** forces immediate escape from the loop, it must jump over the following **THEN**. This results in complications when compiling. A solution to this dilemma is to place the resolving **THEN** into **LOOP** and to implement a conditional compilation which recognizes that a **LEAVE** is being implemented. The resulting construct then becomes:

DO ... IF LEAVE ELSE ... THEN LOOP

The terminating **THEN** is now adjacent to **LOOP** and proper nesting can be maintained.

The accompanying listing is written in "q4th," which differs from fig-FORTH in that it compiles absolute address jumps instead of differential relative jumps. (The listing uses **BACK** to simplify translation.) Also, lowercase words are used instead of parentheses for primitives. The listing uses the fig-FORTH or 79-Standard **LEAVE** which simply sets the loop index equal to the loop limit, and all branching is controlled by **IF...THEN** statements. **WHENLEAVE** and **NOTWHEN** are synonymous and the choice of usage can be decided by the user (or some future standards committee) to make the most sense in context.

It can be seen from the examples in figure two that as soon as the condition for exit is met, then the loop terminates.

It is hoped that the **DO...WHEN...LOOP** construct will be useful and will help resolve some problems with implementing the 83-Standard on older systems.

```
Screen # 17
0 (  DO  WHENLEAVE  WHEN                    rws   11/20/83            )
1 (  fig-FORTH definition      : BACK  HERE - , ;  )
2 (  q4TH definition )         : BACK  , ;
3     : DO    ?COMP  COMPILE do  HERE 3 ;  IMMEDIATE
4     : WHENLEAVE  3 ?PAIRS
5                  [COMPILE] IF
6                  COMPILE LEAVE
7                  [COMPILE] ELSE   ; IMMEDIATE
8
9     : NOTWHEN    [COMPILE] WHENLEAVE  ;   IMMEDIATE
10
11    : WHEN       COMPILE 0=  [COMPILE] WHENLEAVE  ;    IMMEDIATE
12
13
14
15

Screen # 18
0 (  LOOP   +LOOP                           rws   11/20/83            )
1
2     : LOOP   DUP 2 > IF  3 ?PAIRS
3                      ELSE  [COMPILE] THEN
4                      THEN  COMPILE loop  BACK   ;  IMMEDIATE
5
6
7     : +LOOP  DUP 2 > IF  3 ?PAIRS
8                      ELSE HERE 2- DUP @ >R ( save increment  )
9                           2- @ ' literal CFA = ( test if LIT )
10                       IF    -4 ALLOT [COMPILE] THEN
11                                    COMPILE  literal
12                       ELSE  -2 ALLOT [COMPILE] THEN
13                       THEN  R> ,        ( restore increment )
14                     THEN  COMPILE +loop  BACK   ;  IMMEDIATE
15
```

```
DO...WHEN...LOOP          ( or +LOOP )

DO...NOTWHEN...LOOP       ( or +LOOP )

DO...WHENLEAVE...LOOP     ( or +LOOP )
```

Figure One

```
  :COUNTING  20 0 DO  I 15 < WHEN I . LOOP ;
run it
   COUNTING -> 0 1 2 3 4 5 7 8 9 10 11 12 13 14  ok

  :COUNTING-BY-3  30 0 DO  I  21 = NOTWHEN  I .  3 +LOOP ;
run it
   COUNTING-BY-3 -> 0 3 6 9 12 15 18 ok
```

Figure Two

Newton's Method:

Fixed-Point Square Roots in Forth

Nathaniel Grossman
Los Angeles, California

Every specialty has its own folklore. Numerical analysis is no exception, and a big part of this folklore is devoted to tales of one or another calculation that *must* be carried out by such-and-such a method because it can't be pushed through by so-and-so's method in (?)-point. A good example is contained in Klaxon Suralis' article, "Fixed-Point Square Roots" (*Forth Dimensions* IV/1). Before describing a pretty algorithm for extracting sixteen-bit square roots of thirty-two-bit radicands, he asserts that the well-known iterative square-root algorithm called "Newton's Method" is "...best suited to CPUs with full floating-point arithmetic hardware."

This assertion is simply not true. Yes, Newton's method is easier to implement in floating point, but for Forth systems without number-crunching arithmetic hardware, Newton's method is both feasible and fast in fixed point. I remember, as a boy, beginning "high-powered" computation after acquiring a Marchant 9 x 9 x 18 hand-cranked, geared calculator. The arithmetic on that machine was not unlike the formal fixed-point arithmetic in Forth, even up to the lack of a carry digit. I worked through many famous algorithms with that machine nevertheless, and Newton's method was one of them.

I will show here how to implement Newton's method in (fixed-point) Forth, obtaining for any unsigned thirty-two-bit a sixteen-bit, unsigned integer that is the floor of its square root. The core of the method is an observation about iterative solution of equations, that has much greater scope. (For example, it could be used to produce integer cube or higher roots, or the roots of many other equations.)

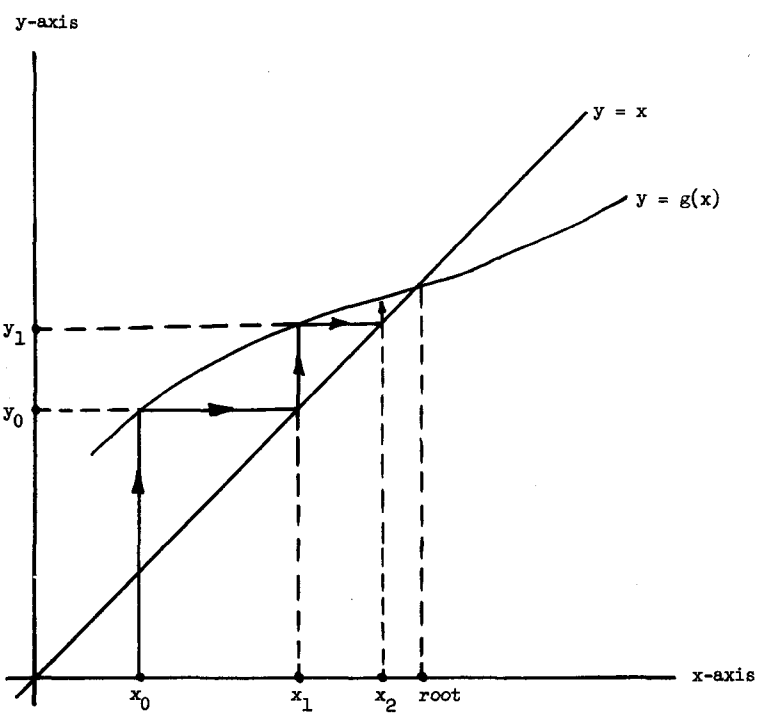


Figure One

Basis for the Algorithm

Suppose M is a positive integer. The goal is the number $[\sqrt{M}]$. The [] notation is standard in mathematics: if x is any real number, then [x] is the largest integer not greater than x. (In the Forth-83 Standard, [x] is called the *floor* of x. For example, [.3] = 0 and [-.4] = -1.)

Newton's method for finding $\sqrt{M}$ is realized in the iteration scheme

$$x_{k+1} = \tfrac{1}{2}(x_k + M/x_k)$$

in which x_0 may be chosen to be any positive number. As k increases, x_k gets closer and closer to $\sqrt{M}$. One can show that this scheme is quadratically convergent, meaning that eventually the number of correct digits doubles with each iteration.

It is obvious that this iteration scheme does not yield only integers. This may be what the folklore is pointing to. But many iteration schemes, and the Newton scheme is one of these, are stable in the sense that small perturbations of the scheme do not destroy the convergence, although it may be slowed down. They can even recover from errors in computation. (Those who carried out hand computation often made copying errors such as transposition of digits, but Newton's method forgives minor trespasses.) We can try to modify Newton's method to work in integers. The modification is easy to explain in a more general setting, and it too is part of the specialized folklore.

A first-order iteration scheme is of the form

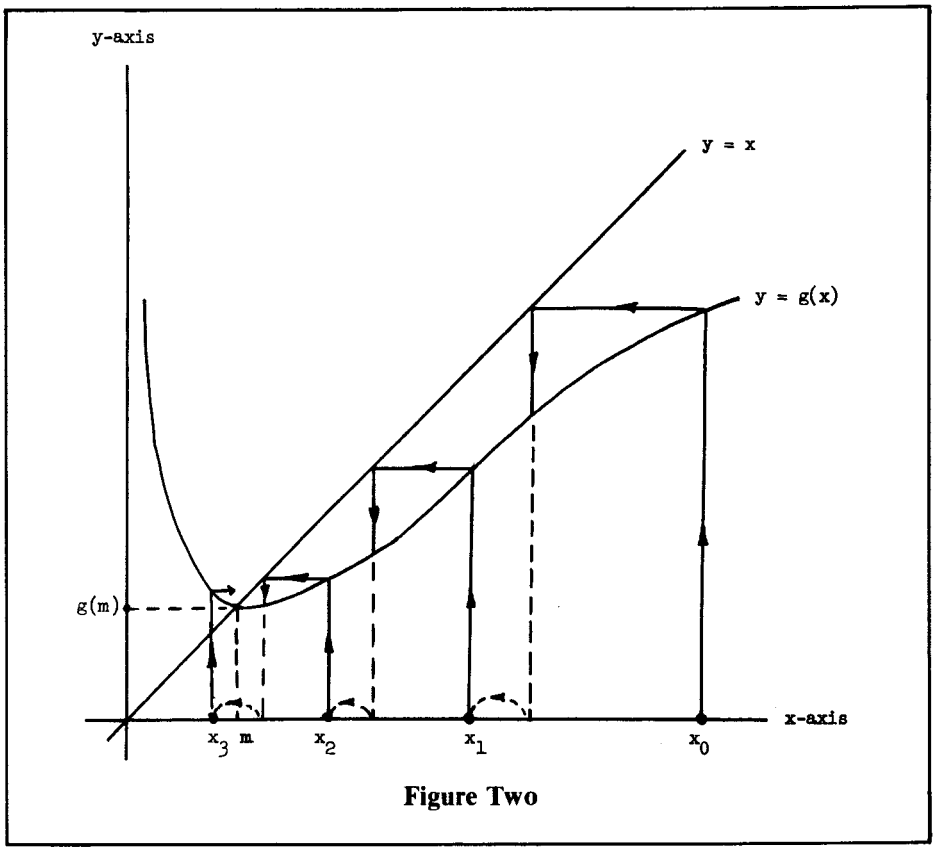


Figure Two

$x_{k+1} = g(x_k)$

with x_0 a suitable choice. If g is "nice" and x_0 is a congenial choice for the starting guess, then x_k gets closer and closer to a root of the equation x = g(x) as k increases. (Different x_0 may lead to different roots.) For example, the function $g(x) = \frac{1}{2}(x + M/x)$ generates Newton's method for square roots, and the positive root of $x = \frac{1}{2}(x + M/x)$ is $\sqrt{M}$.

Each iteration scheme has a geometrical picture. If g(x) generates the scheme, graph the line y = x and the curve y = g(x) on the same axes. Let $y_k = g(x_k)$. Each time an x_k is generated, the corresponding y_k is used for the next x, that is, x_{k+1}. The passage from x to y is made by horizontal and vertical motions broken at the line. Of course, we are seeking the intersection(s) of the line and the curve. Figure one shows the iteration as a flow in a good case.

Suppose that the goal is not the root r for which r = g(r) but its floor [r]. Then we modify the iteration scheme into the form $x_{k+1} = [g(x_k)]$. Now the values of x_k may no longer converge to a limiting value. In some cases, the new x_k may converge to the *integer part* of a root of the equation x = g(x). One case is as follows.

Suppose that g(x) is defined and continuous for x > 0 and that there is an m > 0 so that g(m) is the minimum value of g(x) for x > 0, while g is strictly increasing as x leaves m in either direction. Suppose also that m is the unique root of the equation x = g(x). Pick any $x_0 > m$ and compute x_1, x_2, ... by the scheme $x_{k+1} = [g(x_k)]$. Let k = j be the first index at which the iterates stop marching to the left: $x_j \leq x_{j+1}$ but $x_{k+1} \leq x_k$ for k < j. Then $[m] = x_j$.

Verification of the Modified Scheme

A picture (figure two) helps in following the argument.

Select $x_0 > m$. The ordinate to the curve y = g(x) at $x = x_0$ has value $y_0 = g(x_0)$. But "x_1" = y_0, the intersection of the horizontal at height y_0 with the line y = x. If "x_1" is not an integer, the new scheme requires a hop left to $x_1 = [\text{"}x_1\text{"}]$. This step is iterated to generate x_2, x_3, ... So long as the points (x_k, y_k) be on that branch of y = g(x) to the right of x = m, the iterates will either march to the left or eventually (if m is an integer) remain fixed. In the second case, there will be a first index j such that $x_j = [m] = m$. If the iterates are not eventually fixed, then there must be a k so that $x_k - m < 1$: if not, all k lead to $x_k \geq m + 1$ and the x_k must converge to $p \geq m$. Because g is strictly increasing at x = p and m is a minimum point for g, [g(p)] < m and $x_{k+1} = [g(x_k)] < m < p$ for some k. This is a contradiction; hence there is a first j so that $x_j < m < x_{j-1} < x_{j-2} < \ldots < x_0$ while $x_j < x_{j+1}$. Finally, $x_{j+1} - x_j \leq 1$ because otherwise the horizontal at jth height, y_{j-1}, would intersect y = x at an $x^* > m$. Either this x^* is an integer, contradicting the definition of j, or it is not an integer, in which case $x_j \leq m < x^*$ and $x_j = [m]$. This is the conclusion sought.

Implementation of the Modified Scheme

If $g(x) = (x + M/x)/2$, the generator of Newton's square-root method, the unique positive minimum of g occurs at $x = \sqrt{M}$, the sole positive root of x = g(x). The continuity and monotonicity conditions are satisfied. Thus, the following scheme generates $[\sqrt{M}]$:

Pick $x_0 > \sqrt{M}$ and generate x_1, x_2, ... by

$x_{k+1} = [(x_k + M/x_k)/2]$

If j is the smallest index for which $x_j \leq x_{j+1}$ then $x_j = [\sqrt{M}]$

It is this scheme — or, rather, a slightly fudged version of it — that I implement.

Why is there a fudge? Suppose, to be concrete, that M fits into a thirty-two-bit register while $[\sqrt{M}]$ is to fit into a sixteen-bit register. The arithmetic is to be fixed-point with no carry bits. It may happen either that M/x_k is wider than sixteen-bit or that $x_k + M/x_k$ is wider than sixteen-bit even though the summands are not. The second case might be handled by computing $(x_k/2) + (M/2x_k)$ instead of $(x_k + M/x_k)/2$, but some care would be necessary because $[(x + y)/2] \geq [x/2] + [y/2]$ and > will occur if both x and y are odd integers. Blind implementation of the iteration scheme can (and will) lead to nonsense results and, perhaps, a crash. (This manifests the nasty reality

that the objects in the arithmetic registers are not numbers but only surrogates for numbers, and the arithmetic of those objects only mimics the arithmetic of numbers.) Therefore, the algorithm must be implemented to avoid attempting directly the taking of square roots of numbers that are "too wide." A little experimentation suggests that 2^{30} is already too wide for a thirty-two-bit register.

To create some leeway, suppose that $2^{26} \leq M < 2^{32}$. Write by division $M = 64A + B$, where the quotient A satisfies $2^{20} \leq A < 2^{26}$ and the remainder B satisfies $0 \leq B < 63$. Then A is not too wide, and $[\sqrt{A}]$ can be computed by the Newton scheme without overflow problems.

Note that

$$M = (8[\sqrt{A}])^2 + B + 64A - 64[\sqrt{A}]^2$$

There is an approximation formula

$$\sqrt{(p^2 + q)} \cong p + q/2p$$

We may call this the Babylonian approximation because study of the cuneiform tablet Plimpton 322 (ca. 1900 - 1600 B.C.), found during excavations of Babylon, shows it to be a table of various ratios of sides in right triangles, evidently computed using this approximation formula. The approximation is evidently the x_1 generated by the Newton formula when $x_0 = p$ is chosen. The approximation also represents the first two terms in the infinite series for $\sqrt{(p^2 + q)}$ when the binomial theorem for fractional exponents, first written out by Isaac Newton, is employed. Application of the Babylonian formula to the expression for M with $p = 8[\sqrt{A}]$ and $q = B + 64A - 64[\sqrt{A}]^2$ gives the approximation hereafter referred to as (*):

$$\sqrt{M} \cong 8[\sqrt{A}] + (B + 64A - 64[\sqrt{A}]^2)/16[\sqrt{A}]$$

We will implement the formula (*) in fixed-point Forth. The first term, $8[\sqrt{A}]$, is an integer. The second term is not necessarily an integer, but the integer part of $\sqrt{M}$ is the sum of $8[\sqrt{A}]$ with the integer part of the second term. To be sure, it is necessary to rule out the possibility that additional correction terms will take the refined

```
 0 ( SCR #1: 16-BIT SQRT OF 32-BIT INTEGERS           )
 1 ( NATHANIEL GROSSMAN, 9/27/83 --- HES VIC-FORTH    )
 2 ( DOUBLE PRECISION MATHEMATICS WORDS, AFTER FD-V,1 )
 3: T*       ( UD,UN --- UT )
 4   DUP ROT U* >R >R
 5       U*
 6   0 R> R> D+ ;
 7: T/       ( UT,UN --- UD )
 8    >R  R  U/ SWAP
 9   ROT 0 R U/ SWAP
10   ROT  R> U/ SWAP DROP
11   0 2SWAP SWAP D+ ;
12: U*/     ( UD,UN,UN --- UD )
13   >R T* R> T/ ; ;S
14
15


 0 ( SCR #2: 16-BIT SQRT, CONT  NG,9/27/83 )
 1: D<    ( AFTER "ALL ABOUT FORTH" )
 2   ROT 2DUP =
 3   IF      ROT ROT DMINUS D+
 4   ELSE    SWAP < SWAP DROP
 5   ENDIF   SWAP DROP ;
 6: DU<   ( FD-IV,1 )
 7   32768 + ROT 32768 + ROT ROT D< ;
 8
 9    0 VARIABLE #DYADS
10 1024 CONSTANT ROOT-CUT
11
12
13
14
15


 0 ( SCR #3: 16-BIT SQRT, CONT  NG,9/27/83 )
 1: DYADS-IN-CELL?   ( UN --- )
 2   BEGIN
 3    4 / DUP                 ( SHIFT BY DYADS TO THE RIGHT    )
 4   WHILE                    ( IF NOT SHIFTED INTO 0          )
 5    1 #DYADS +!             ( THEN INCREASE #DYADS BY 1      )
 6   REPEAT DROP ;
 7: DYAD-COUNT       ( UD --- UD )
 8   1 #DYADS ! 2DUP          ( INITIALIZE                     )
 9    IF     8 #DYADS +!      ( IF HIGH PART NOT 0, INCREASE   )
10           DROP DUP         ( #DYADS BY 8, DROP LOW PART     )
11           DYADS-IN-CELL?   ( COUNT DYADS                    )
12    ELSE   DYADS-IN-CELL?   ( IF HIGH PART IS 0, COUNT DYADS )
13    ENDIF   ; ;S
14
15


 0 ( SCR #4: 16-BIT SQRT, CONT  NG,9/27/83 )
 1: FIRST-GUESS    ( SETS UP UN 111111 ... 11 )
 2   DYAD-COUNT        ( COUNT DYADS                          )
 3   #DYADS @ 1 -
 4   DUP 0 =           ( ONLY ONE DYAD?                       )
 5    IF               ( YES?                                 )
 6     DROP 2          ( FIRST GUESS IS 2                     )
 7    ELSE             ( MORE THAN ONE DYAD?                  )
 8     1 SWAP          ( THEN FILL OUT THE FIRST GUESS        )
 9     0 DO            ( WITH BINARY DIGITS 1                 )
10        DUP + 1 +
11       LOOP
12    ENDIF ; ;S
13
14
15
```

```
 0 ( SCR #5: 16-BIT SQRT, CONT  NG,9/27/83 )
 1: HERON   ( UD,UN --- UD,UN )
 2   BEGIN                          ( START ITERATION LOOP      )
 3    >R 2DUP R                     ( SET UP DATA               )
 4    U/ SWAP DROP S->D R S->D      ( DIV BY TRIAL ROOT, TURN   )
 5    D+ 2 U/ SWAP DROP             (  GUESS AND QUOT INTO DBL  )
 6    R>                            (  AVERAGE, AND SAVE NEW    )
 7    2DUP <                        ( OLD < NEW ?               )
 8     IF          DROP 0           ( YES? DROP NEW, EXIT LOOP  )
 9     ELSE   SWAP DROP 1           ( NO? GO AROUND AGAIN       )
10     ENDIF
11   UNTIL ;
12: NEWTON-[SQRT]  ( UD,UN --- UN )
13   FIRST-GUESS
14   HERON
15   SWAP DROP SWAP DROP ; ;S


 0 ( SCR #6: 16-BIT SQRT, CONT  NG,9/27/83 )
 1: RAD(MOD64)      ( UD --- UN,UD; REM,QUOT MOD 64 )
 2   2DUP
 3   1 64 U*/               ( DIV BY 64 TO GET A                )
 4   2 DUP >R >R            ( GET COPY OF QUOT                  )
 5   64 1 U*/               ( GET                               )
 6   DMINUS D+              (  REMAINDER                        )
 7   DROP R> R> ;           ( CONVERT D TO S, RECALL QUOT       )
 8: CORRECTED- SQRT ( N,N,UN --- UN )
 9   DUP 2DUP >R >R         ( MAKE, STORE COPIES OF [SQRT(A)]   )
10   U* DMINUS D+           ( A - [SQRT(A)]**2                  )
11   64 1 U*/               ( GET                               )
12   ROT S->D D+            (  NUMERATOR                        )
13   R> U/ 16 /             ( DIV TO GET CORRECTION             )
14   R> 8 * + ; ;S          ( MAIN TERM PLUS CORRECTION         )
15


 0 ( SCR#7: 16-BIT SQRT, CONCLUDED  NG,9/27/83)
 1: SQRT     ( UD --- UN )
 2   2DUP
 3   0 ROOT-CUT DU<       ( IS RADICAND SMALL?                  )
 4   IF                   ( YES? THEN                           )
 5    NEWTON-[SQRT]       (  TAKE THE ROOT DIRECTLY             )
 6   ELSE                 ( NO? THEN TAKE IT INDIRECTLY         )
 7    RAD(MOD64)          ( GET REDUCED RADICAND                )
 8    2DUP                (  AND TAKE ITS SQRT                  )
 9    NEWTON-[SQRT]       ( GET CORRECTION, BLOW UP             )
10    CORRECTED-SQRT      (  REDUCED ROOT, AND ADD              )
11   ENDIF ; ;S
12
13  ( KEY IN A 32-BIT -- THAT IS, UNSIGNED DOUBLE -- )
14  ( THEN SQRT, AND EXECUTE. RESULT IS 16-BIT THAT  )
15  ( IS FLOOR OF SQRT OF 32-BIT.                    )
```

approximation across an integer, altering the value assigned to $[\sqrt{M}]$.

The following worst-possible-case estimate shows that additional correction terms will not change the value of $[\sqrt{M}]$ given by the two-term Babylonian approximation. By examining the rest of the terms in the infinite series for $\sqrt{(p^2 + q)}$ one can show that the total contribution from the infinitely many terms neglected by the two-term approximation (*) for M is no bigger in absolute value than

$$\frac{q^2}{8\,p^3} = \frac{(B + 64A - 64[\sqrt{A}])^2}{2^{12}[\sqrt{A}]^3}$$

and in fact is negative. Write $\sqrt{A} = [\sqrt{A}] + \delta$, $0 \le \delta < 1$. Then $A = [\sqrt{A}]^2 = [\sqrt{A}]^2 + 2\delta[\sqrt{A}] + \delta^2$, so that $B + 64A - 64[\sqrt{A}]^2 = B + 128\delta[\sqrt{A}] + 64\delta^2$. But, $0 \le B < 63$ and both $\delta < 1$ and $\delta^2 < 1$, so that $B + 128\delta[\sqrt{A}] + 64\delta^2 < 128[\sqrt{A}] + 128 \le 256[\sqrt{A}]$. Therefore, the magnitude of the secondary correction can be no more than

$$\frac{(256[\sqrt{A}])^2}{2^{12}[\sqrt{A}]^3} = \frac{16}{[\sqrt{A}]} < \frac{16}{2^{10}} = \frac{1}{64}$$

On the other hand, if the primary correction is not 0, the ratio of the corrections is

$$\frac{\text{secondary correction}}{\text{primary correction}} = \frac{(B + \ldots)^2}{4096[\sqrt{A}]^3} \times \frac{16[\sqrt{A}]}{(B + \ldots)} = \frac{(B + \ldots)}{256[\sqrt{A}]^2} < 1$$

Hence, the contribution from the primary correction cannot be canceled out by the negative secondary correction, and (*) is satisfactory.

Coding the Algorithm into Forth

I implemented an algorithm based upon (*) on a VIC-20 computer operating with HES VIC-FORTH (which has the **U/** bug described in *Forth Dimensions* IV and V). The algorithm accepts thirty-two-bit radicands and produces sixteen-bit square roots. Note that the algorithm is extensible to wider radicands, but such extensions will require division to be extended in order that divisors wider than sixteen bits be accepted. If possible time penalties coming from division extensions be ignored, the running time for a Newton-style square root on a radicand b bits wide will be proportional to log b. The time to run a square root algorithm of Suralis' style will be proportional to b, so the Newton-type will eventually be far superior.

Here is a commentary on the accompanying Forth screens:

Screen #1: The words **T***, **T/** and **U*/** are taken from *Forth Dimensions* (V/1). I tried to modify them to return remainder as well as quotient along the lines suggested by Bieman, but ran into some difficulties that may come from the 6502 **U/** bug built into HES VIC-FORTH.

Screen #2: The words **D<** and **DU<** follow Suralis (*Forth Dimensions* IV/1).

Screen #3: The choice of initial square root approximation is crucial in keeping down running time. If the first guess is far away from the true square root, many extra time-costly iterations

(with their divisions) will be necessary. On the other hand, the method requires that the initial guess be no smaller than the true square root. Machines using floating point typically compute the first guess from a rational function of the radicand, and this function is derived by Pade approximation or in some other esoteric way. Fixed-point computers do not enjoy such freedom. I obtain an optimal first guess by counting dyads. The radicand M is represented by a binary integer with leading digit 1. Starting from the right, mark off dyads — pairs of digits — and count how many dyads, say p, are needed to cover M. Then [M] will be an integer of p binary digits, the leftmost 1. The largest integer with p binary digits is just 111...11, a string of p 1s, and this I take for the first guess at [M]. Since 111...11 itself is the square root of its square, the choice is optimal.

The word **DYADS-IN-CELL?** counts dyads in a sixteen-bit number. **DYAD-COUNT** counts dyads in a thirty-two-bit number: if the high cell is non-zero, the counting storehouse **#DYADS** is credited with eight dyads for the low cell and the high cell count is added in, while if the high point is zero, it is dropped and the low cell is counted.

Screen #4: The word **FIRST-GUESS** sets up the first guess as a line of 1s according to the scheme explained under Screen #3.

Screen #5: The Russian commentators credit the Greek mathematician Heron of Alexandria (second half of the first century A.D.) with inventing the iteration scheme used for Newton-style square roots. As I have remarked above, the basic method was known much earlier to the Babylonians. But HERON is much shorter than BABYLONIAN, so **HERON** is the word that carries out the iteration. Then **NEWTON-[SQRT]** combines the **FIRST-GUESS** and the **HERON** iteration to give the integer part [M] if M is not "too wide." This word does not check width, so it will return nonsense if used on a radicand that is too wide.

Screen #6: The word **RAD(MOD64)** decomposes a thirty-two-bit radicand into the form M = 64A + B.

Screen #7: The word **SQRT** checks width first. If M is not too wide, **SQRT** takes the square directly. Otherwise, **SQRT** decomposes M, takes the square root of A, computes the correction, and assembles the square root of M using the word **CORRECTED-SQRT** from screen #6.

Fig Chapters

U.S.

• ARIZONA

Phoenix Chapter
Call Dennis L. Wilson
602/956-7678

• CALIFORNIA

Los Angeles Chapter
Monthly, 4th Sat., 11 a.m.
Allstate Savings
8800 So. Sepulveda Boulevard
1/2 mile North of LAX
Los Angeles
Call Phillip Wasson
213/649-1428

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

Orange County Chapter
Monthly, 4th Wed., 7 p.m.
Fullerton Savings
Talbert & Brookhurst
Fountain Valley
Monthly, 1st Wed., 7 p.m.
Mercury Savings
Beach Blvd. & Eddington
Huntington Beach
Call Noshir Jesung
714/842-3032

San Diego Chapter
Weekly, Thurs., 12 noon.
Call Guy Kelly
619/268-3100 ext. 4784

• COLORADO

Denver Chapter
Monthly, 1st Mon., 7 p.m.
Call Steven Sarns
303/477-5955

• ILLINOIS

Fox Valley Chapter
Call Samuel J. Cook
312/879-3242

Rockwell Chicago Chapter
Call Gerard Kusiolek
312/885-8092

• INDIANA

Central Indiana Chapter
Monthly, 3rd Sat.
Call Richard Turpin
317/923-1321

• IOWA

Iowa City Chapter
Monthly, 4th Tues.
Engineering Bldg., Rm. 2128
University of Iowa
Call Robert Benedict
319/337-7853

• KANSAS

Wichita Chapter (FIGPAC)
Monthly, 3rd Wed., 7 p.m.
Wilber E. Walker Co.
532 S. Market
Wichita, KS
Call Arne Flones
316/267-8852

• MASSACHUSETTS

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

• MINNESOTA

MNFIG Chapter
Even month, 1st Mon., 7:30 p.m.
Odd Month, 1st Sat., 9:30 a.m.
Vincent Hall, Univ. of MN
St. Paul, MN
Call Fred Olson
612/588-9532

• MISSOURI

Kansas City Chapter
Monthly, 4th Tues., 7 p.m.
Midwest Research Inst.
Mag Conference Center
Call Linus Orth
816/444-6655

St. Louis Chapter
Monthly, 3rd Tue., 7 p.m.
Thornhill Branch of
St. Louis County Library
Call David Doudna
314/867-4482

• NEVADA

Southern Nevada Chapter
Suite 900
101 Convention Center Drive
Las Vegas, NV
Call Gerald Hasty
702/452-3368

• NEW JERSEY

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

• NEW YORK

New York Chapter
Monthly, 2nd Wed., 8 p.m.
Queens College
Call Tom Jung
212/432-1414 ext. 157 days
212/261-3213 eves.

Rochester Chapter
Bi-monthly, 4th Sat., 2 p.m.
March, May & June
Hutchison Hall
Univ. of Rochester
Call Thea Martin
716/235-0168

Syracuse Chapter
Monthly, 1st Tues., 7:30 p.m.
Call C. Richard Corner
315/456-7436

• OHIO

Athens Chapter
Call Isreal Urieli
614/594-3731

Cleveland Chapter
Call Gary Bergstrom
216/247-2492

Dayton Chapter
Twice monthly, 2nd Tues & 4th Wed., 6:30 p.m.
CFC, 11 W. Monument Ave.
Suite 612
Dayton, OH
Call Gary M. Granger
513/849-1483

• OKLAHOMA

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

• OREGON

Greater Oregon Chapter
Monthly, 2nd Sat., 1 p.m.
Computer & Things
3460 SW 185th, Aloha
Call Timothy Huang
503/289-9135

• PENNSYLVANIA

Philadelphia Chapter
Monthly, 3rd Sat.
LaSalle College, Science Bldg.
Call Lee Hustead
215/539-7989

• TEXAS

Dallas/Ft. Worth Metroplex Chapter
Monthly, 4th Thurs., 7 p.m.
Software Automation, Inc.
14333 Porton, Dallas
Call Chuck Durrett
214/788-1655
Bill Drissel
214/264-9680

Houston Chapter
Call Dr. Joseph Baldwin
713/749-2120

• VERMONT

Vermont Fig Chapter
Monthly, 3rd Mon., 7:30 p.m.
Vergennes Union High School
Rm. 210, Monkton Rd.
Vergennes, VT
Call Hal Clark
802/877-2911 days
802/452-4442 eves

• VIRGINIA

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

Richmond Forth Group
Monthly, 2nd Wed., 7 p.m.
Basement, Puryear Hall
Univ. of Richmond
Call Donald A. Full
804/739-3623

FOREIGN

• AUSTRALIA

Melbourne Chapter
Monthly, 1st Fri., 8 p.m.
Contact: Lance Collins
65 Martin Road
Glen Iris, Victoria 3146
03/29-2600

Sydney Chapter
Monthly, 2nd Fri., 7 p.m.
John Goodsell Bldg.,
Rm LG19
Univ. of New South Wales
Sydney
Contact: Peter Tregeagle
10 Binda Rd., Yowie Bay
02/524-7490

Forth Times Fig Chapter
Contact: Ritchie Laird
25 Gibsons Road
Sale, Victoria 3850
051/44-3445

• BELGIUM

Belgium Chapter
Monthly, 4th Wed., 20:00h
Contact: Luk Van Loock
Lariksdreff 20
2120 Schoten
03/658-6343

• CANADA

Nova Scotia Chapter
Contact: Howard Harawitz
227 Ridge Valley Rd.
Halifax, Nova Scotia B3P 2E5
902/542-7812

Southern Ontario Chapter
Monthly, 1st Sat., 2 p.m.
General Sciences Bldg,
Rm 312
McMaster University
Contact: Dr. N. Solntseff
Unit for Computer Science
McMaster University
Hamilton, Ontario L8S 4K1
416/525-9140 ext. 2065

• COLOMBIA

Colombia Chapter
Contact: Luis Javier Parra B.
Aptdo. Aereo 100394
Bogota
214-0345

• ENGLAND

Forth Interest Group — U.K.
Monthly, 1st Thurs., 7 p.m
Bradden Old Rectory
Towchester, Northamptonshire
NN12 8ED
Contact: Keith Goldie-Morrison
15 St. Albans Mansion
Kensington Court Place
London W8 5QH

• FRANCE

French Language Chapter
Contact: Jean-Daniel Dodin
77 rue du Cagire
31100 Toulouse
(16-61) 44.03.06

• IRELAND

Irish Chapter
Contact: Hugh Dobbs
Newton School
Waterford
051/75757
051/74124

• ITALY

FIG Italia
Contact: Marco Tausel
Via Gerolamo Forni 48
20161 Milano
02/645-8688

• SWITZERLAND

Swiss Chapter
Contact: Max Hugelshofer
ERNI & Co. Elektro-Industrie
Stationsstrasse
8306 Bruttisellen
01/833-3333

• TAIWAN

Taiwan Chapter
Contact: J.N. Tsou
Forth Information Technology
P.O. Box 53-200
Taipei
02/331-1316

SPECIAL GROUPS

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

Baton Rouge Atari Chapter
Call Chris Zielewski
504/292-1910

Detroit Atari Chapter
Monthly, 4th Wed.
Call Tom Chrapkiewicz
313/524-2100

FIGGRAPH
Call Howard Pearlmutter
408/425-8700

Letters *(Continued from page 12)*

once or twice. This results in an exponential or hyper-exponential distribution. The exact nature of the distribution can be used to design fairly general compacting codes with savings of two to four over the already very compact address strings. (Of course, the address interpreter becomes slower and larger.)

James C. Brakefield
5803 Cayuga
San Antonio, TX 78228

Variable Urges

Hello:

I'm a Forth novice, and I just read Michael Ham's "Why Novices Use So Many Variables" (*Forth Dimensions* V/4) and I should like to add to what he says.

My own urges to define variables are rooted in my experience with other languages. I am accustomed to saying things like X = X + 1 and INC X. Each number used in non-Forth programs has been a static entity whose value could be evoked by merely mentioning its name. But numbers in Forth are most powerfully managed by keeping them in a stack where they may be accessed rapidly. And the stack elements are dynamic entities — this is where my problems arise.

In Forth, I am required to prepare a strategy by which each value on the stack will be in the right place at the right time. This is a nuisance. What was once taken care of by an assignment statement now requires thought: "How can I make this number work on the stack without messing up the code by including stack words whose operands are difficult to determine?" And there's the rub. Among Forth programming's other unusual attributes, there is the necessary activity of planning when and where a number will be used.

Without malice a' Forth-thought,

Bryn Aash
4601 S.W. 58th Ave.
Miami, FL 33155

CQ...CQ...

Dear Sir:

After having purchased a new computer, I wish to interface it to a ham radio for the purpose of copying CW and RTTY. The computer is Epson's QX-10, and is set up to handle CP/M-80, MBASIC and Z80 FORTH by Laboratory Microsystems.

Whatever help your company can give me in getting this interface going will please me and a few of my friends. If this configuration doesn't work, I have been looking to purchase the Kaypro 4, if it will interface. As far as software control, I do need a good program that will produce ASCII files that my word processor can read.

Thank you,

Glen A. Fuller
WA8EQO
Box 765 USCG S/C
Kodiak, AK 99619

(Continued on page 38)

Techniques Tutorial

Self-Defining Words

Henry Laxen
Berkeley, California

Before I get into the topic of what I call self-defining words, I would like to give a little bit of "motivation," as mathematicians put it when they are ashamed to admit they are just making things up. Let's start with execution vectors or, as I like to call them, deferred words. Consider the definition in figure one, which I am sure you will recognize.

DEFER is a defining word which allows you to change its run-time definition at a later time by simply storing a new code field address into the parameter field of the word defined by **DEFER**. (I am assuming you are using an 83-Standard Forth system.)

For example, suppose we define the words in figure two. Now by setting the parameter field of **HELLO** to be either the code field address of **RUSSIAN** or the code field address of **GERMAN**, we can change the execution behavior of **GREETING** without modifying **GREETING** at all. This ability to change the run-time behavior of words is very powerful, and has been discussed at length in my article on execution vectors (*Forth Dimensions* III/6).

Suppose we want to do more than just defer the run-time behavior of a word. For example, consider the top-of-page function for a printer. It consists of two components, one that is printer dependent and another that is printer independent. If we have system variables called **PAGE#** and **#LINES** then every time we want a new page on the printer, we need to execute code which will cause the printer to advance to the top of the next page, as well as increment **PAGE#** and set **#LINES** back to 0. If we do this with the **DEFER** mechanism, we would have to do something like that shown in figure three.

That is not too bad, but there is no real reason to have a definition called **TOP-OF-FORM**. (I try to be frugal with the number of names I add to my system, since I am approaching thirty and my memory is beginning to fail.) Another approach is to create a custom defining word, as in figure four.

Again, this is a poor solution, since the **PAGE:** defining word is basically wasted. It is very unlikely to be used again to define another version of **PAGE**.

At this point, I was forced to be creative, and I thought, "Why do **CREATE** and **DOES>** always have to be together?" The answer is, they don't. In fact, we often use **CREATE** without **DOES>** when we define pointers or arrays. What would happen if we used **DOES>** without a **CREATE**? Suppose we did the deed expressed in figure five?

Well, unfortunately, this is implementation dependent, but on most Forth-79 and Forth-83 systems it should work as follows. You must execute **PAGE** immediately, before defining any other words. This will cause the code field of **PAGE** to be re-written to point to the **DOES>** portion of the definition. At this point, you must set the parameter field of **PAGE** to the code field of the word you want to execute to cause the paper to advance. On a Forth-83 system, the code in figure six would work.

From now on, when you execute **PAGE**, then **FORM-FEED** will be executed, followed by the **1 PAGE# +!** and the **0 #LINES !** is next. If you change printers and need to modify how **PAGE** works, then you need only set the parameter field of **PAGE** to a different code field. Now the only question is, what in the world are we doing, and why the hell does it work? The answer lies in the implementation of **DOES>**.

On Forth-79 and Forth-83 systems, **DOES>** is an immediate word, which compiles a word called **(;CODE)** followed by a call instruction to the run-

Figure One

```
: DEFER    CREATE ['] ABORT ,  DOES> @  EXECUTE  ;
```

Figure Two

```
: RUSSIAN    ." Zdrasvoytyeh"   ;
: GERMAN     ." Guten Tag"   ;
DEFER HELLO
: GREETING ." The only foreign greeting I know is " HELLO ;
```

Figure Three

```
DEFER TOP-OF-FORM

: PAGE  TOP-OF-FORM  1 PAGE# +!  0 #LINES !  ;
```

Figure Four

```
: PAGE:  VARIABLE  DOES> @ EXECUTE  1 PAGE# +!  0 #LINES !  ;
PAGE: PAGE
```

Figure Five

```
: PAGE  DOES> @ EXECUTE  1 PAGE# +!  0 #LINES !  ;
```

Figure Six

```
: FORM-FEED  12 EMIT  ;
' FORM-FEED  ' PAGE >BODY !
```

Figure Seven

```
: (;CODE)  R> LAST @ NAME> !  ;
```

time code for **DOES>**. The run-time code pushes the address of the parameter field of the current word onto the parameter stack, and causes interpretation to proceed at the words following the **DOES>** in the definition. The **(;CODE)** word causes the code field of the most recently defined definition to be re-written to point to the call instruction that was compiled by **DOES>**. Let's look at what is compiled by the above example:

Header of **PAGE** followed by run-time for : usually called **NEST**

cfa of **(;CODE)**

CALL DODOES> A machine-language call to the run-time for **DOES>**

cfa of @

cfa of **EXECUTE**

etc.

When **PAGE** is executed the first time, **(;CODE)** is executed. Its definition is something like that shown in figure seven, where **LAST** is a variable that points to the name field of the most recently defined definition, and **NAME>** is an operator that converts a name field to a code field. In essence, **(;CODE)** re-writes the code field of the most recently defined word to point to the byte immediately following the **(;CODE)** code field. Thus, in the above definition of **PAGE**, after it is executed for the first time the code field of **PAGE** will point to the **CALL DODOES>** instruction that was compiled by **DOES>**. Also, by coincidence or foresight, the first two bytes of the parameter field of **PAGE**, which contain the code field of **(;CODE)**, are now available for something else. Thus, if you set the parameter field of **PAGE** to the code field of the word you want to execute via the **@ EXECUTE** clause, it will be done. By executing **PAGE** once, it has re-defined itself and, in fact, has become an execution vector plus something extra, namely the page incrementing and line number re-setting part. This is why I call the above construct a self-defining word, since it has the ability to re-define itself.

Now for the admission of guilt. The fact is, that like a mathematician, the way I came up with self-defining words was by just playing around. The motivation was conceived of afterward as a justification for using the crazy scheme I have described above. Having come up with this strange mechanism that happened to work because of a comedy of circumstances, it took me quite a while to dream up an example in which it would be useful. However, the **PAGE** example is valid and, having discovered the technique, it is not uncommon to find an application for it. I hope you will be able to use it in useful ways, besides just confusing your Forth friends.

Until next time, good luck, and may the Forth be with you!

THE FORTH SOURCE™

MVP-FORTH

Stable - Transportable - Public Domain - Tools

You need two primary features in a software development package . . . a stable operating system and the ability to move programs easily and quickly to a variety of computers. MVP-FORTH gives you both these features and many extras. This public domain product includes an editor, FORTH assembler, tools, utilities and the vocabulary for the best selling book "Starting FORTH". The Programmer's Kit provides a complete FORTH for a number of computers. Other MVP-FORTH products will simplify the development of your applications.

MVP Books - A Series

- ☐ **Volume 1,** *All about FORTH* by Haydon. MVP-FORTH glossary with cross references to fig-FORTH, *Starting FORTH* and FORTH-79 Standard. 2nd Ed. **$25**
- ☐ **Volume 2,** *MVP-FORTH Assembly Source Code.* Includes CP/M®, IBM-PC®, and APPLE® listing for kernel $20
- NEW ☐ **Volume 3,** *Floating Point Glossary* by Springer $10
- NEW ☐ **Volume 4,** *Expert System* with source code by Park $25
- NEW ☐ **Volume 5,** *File Management System* with interrupt security by Moreton $25

MVP-FORTH Software - A Transportable FORTH

- ☐ **MVP-FORTH Programmer's Kit** including disk, documentation, Volumes 1 & 2 of MVP-FORTH Series (*All About FORTH*, 2nd Ed. & Assembly Source Code), and *Starting FORTH*. Specify ☐ CP/M, ☐ CP/M 86, ☐ CP/M+, ☐ APPLE, ☐ IBM PC, ☐ MS-DOS, ☐ Osborne, ☐ Kaypro, ☐ H89/Z89, ☐ Z100, ☐ TI-PC, ☐ MicroDecisions, ☐ Northstar,
- NEW ☐ **Compupro, ☐ Cromenco, ☐ DEC Rainbow, ☐ NEC 8201,** ☐ TRS-80/100 $150
- ☐ **MVP-FORTH Cross Compiler** for CP/M Programmer's Kit. Generates headerless code for ROM or target CPU $300
- ☐ **MVP-FORTH Meta Compiler** for CP/M Programmer's kit. Use for applicatons on CP/M based computer. Includes public domain source $150
- ☐ **MVP-FORTH Fast Floating Point** Includes 9511 math chip on board with disks, documentation and enhanced virtual MVP-FORTH for Apple II, II+, and IIe. **$450**
- ☐ **MVP-FORTH Programming Aids** for CP/M, IBM or APPLE Programmer's Kit. Extremely useful tool for decompiling, callfinding, and translating. $150
- ☐ **MVP-FORTH PADS (Professional Application Development System)** for IBM, or APPLE. An integrated development system with complete documentation for PC, XT or PCjr and Apple II, II+ and IIe. Will run on most IBM and Apple look-alikes. $500
- NEW ☐ **MVP-FORTH Floating Point & Matrix Math** for IBM or Apple $85
- NEW ☐ **MVP-FORTH Graphics Extension** for IBM or Apple $65
- NEW ☐ **MVP-FORTH MS-DOS** file interface for IBM PC PADS **$80**
- NEW ☐ **MVP-FORTH Expert System** for development of knowledge-based programs for Apple, IBM, or CP/M. $100

FORTH COMPUTER

- ☐ **Jupiter Ace** $150
  - ☐ 16K RAM Pack $50
  - ☐ 48K RAM Pack $125

FORTH CROSS COMPILERS Allow extending, modifying and compiling for speed and memory savings, can also produce ROMable code. •Requires FORTH disk.

☐ CP/M	$300	☐ IBM•	$300
☐ 8086•	$300	☐ Z80•	$300
☐ 68000	$300	☐ Apple II/II+	$300

Ordering Information: Check, Money Order (payable to MOUNTAIN VIEW PRESS, INC.), VISA, MasterCard, American Express. COD's $5 extra. Minimum order $15. No billing or unpaid PO's. California residents add sales tax. Shipping costs in US included in price. Foreign orders, pay in US funds on US bank, include for handling and shipping by Air: $5 for each item under $25, $10 for each item between $25 and $99 and $20 for each item over $100. All prices and products subject to change or withdrawal without notice. Single system and/or single user license agreement required on some products.

FORTH DISKS

FORTH with editor, assembler, and manual.

- ☐ **APPLE** by MM $100
- ☐ **APPLE** by Kuntze $90
- ☐ **ATARI®** valFORTH $60
- ☐ **CP/M®** by MM $100
- ☐ **HP-85 by Lange** **$90**
- ☐ **HP-75** by Cassady $150
- ☐ **IBM-PC®** by LM $100
- ☐ **NOVA** by CCI 8" DS/DD $175
- ☐ **Z80** by LM $100
- ☐ **8086/88** by LM $100
- ☐ **68000** by LM $250
- ☐ **VIC FORTH** by HES, VIC20 cartridge **$50**
- ☐ **C64** by HES Commodore 64 cartridge $60
- ☐ **Timex** by HW $25

Enhanced FORTH with: F-Floating Point, G-Graphics, T-Tutorial, S-Stand Alone, M-Math Chip Support, MT-Multi-Tasking, X-Other Extras, 79-FORTH-79, 83-FORTH-83.

- ☐ **APPLE** by MM, F, G, & 83 $160
- ☐ **ATARI** by PNS, F,G, & X. $90
- ☐ **CP/M** by MM, F & 83 $160
- ☐ **Apple, GraFORTH** by I $75
- ☐ **Multi-Tasking FORTH** by SL, CP/M, X & 79 $395
- ☐ **TRS-80/I or III** by MMS F, X, & 79 $130
- ☐ **Timex** by FD, tape G,X, & 79 **$45**
- ☐ **Victor 9000** by DE,G,X $150
- ☐ **Extensions** for LM Specify IBM, Z80, or 8086
  - ☐ Software Floating Point $100
  - ☐ 8087 Support (IBM-PC or 8086) $100
  - ☐ 9511 Support (Z80 or 8086) $100
  - ☐ Color Graphics (IBM-PC) $100
  - ☐ Data Base Management $200

  Requires LM FORTH disk.
- ☐ **fig-FORTH Programming Aids** for decompiling, callfinding, and translating. CP/M, IBM-PC, Z80, or Apple $150

FORTH MANUALS, GUIDES & DOCUMENTS

- ☐ **ALL ABOUT FORTH** by Haydon. See above. $25
- ☐ **FORTH Encyclopedia** by Derick & Baker $25
- NEW ☐ **The Complete FORTH** by Winfield $16
- ☐ **Understanding FORTH** by Reymann $3
- ☐ **FORTH Fundamentals,** Vol. I by McCabe $16
- NEW ☐ **FORTH Fundamentals,** Vol. II by McCabe $13
- NEW ☐ **FORTH Tools, Vol.1** by Anderson & Tracy $20
- NEW ☐ **Beginning FORTH** by Chirlian $17
- ☐ **FORTH Encyclopedia Pocket Guide** $7
- ☐ **And So FORTH** by Huang. A college level text. $25
- ☐ **FORTH Programming** by Scanlon $17
- ☐ **FORTH on the ATARI** by E. Floegel $8
- ☐ **Starting FORTH** by Brodie. Best instructional manual available. (soft cover) $18
- ☐ **Starting FORTH** (hard cover) $23
- NEW ☐ **68000** fig-Forth with assembler **$25**
- NEW ☐ **Jupiter** ACE Manual by Vickers $15
- ☐ **1980 FORML Proc.** $25
- ☐ **1981 FORML Proc** 2 Vol $40
- ☐ **1982 FORML Proc.** $25
- ☐ **1981 Rochester FORTH Proc.** $25
- ☐ **1982 Rochester FORTH Proc.** $25
- NEW ☐ **1983 Rochester FORTH Proc.** $25
- ☐ **A Bibliography of FORTH References, 1st. Ed.** $15
- ☐ **The Journal of FORTH Application & Research Vol. 1, No. 1** $20
- ☐ **Threaded Interpretive Languages** $23
- ☐ **METAFORTH** by Cassady $30
- ☐ **Systems Guide to fig-FORTH** $25
- ☐ **Invitation to FORTH** $20
- ☐ **PDP-11 User Man.** $20
- ☐ **FORTH-83 Standard** $15
- ☐ **FORTH-79 Standard** $15
- ☐ **FORTH-79 Standard Conversion** $10
- ☐ **Tiny Pascal fig-FORTH** $10
- ☐ **NOVA fig-FORTH** by CCI Source Listing $25
- ☐ **NOVA** by CCI User's Manual $25
- ☐ **Installation Manual for fig-FORTH,** $15

Source Listings of fig-FORTH, for specific CPU's and computers. The Installation Manual is required for implementation. Each $15

☐ 1802	☐ 6502	☐ 6800	☐ AlphaMicro
☐ 8080	☐ 8086/88	☐ 9900	☐ APPLE II
☐ PACE	☐ 6809	☐ NOVA	☐ PDP-11/LSI-11
☐ 68000	☐ Eclipse	☐ VAX	☐ Z80

MOUNTAIN VIEW PRESS, INC.

PO BOX 4656 **MOUNTAIN VIEW, CA 94040** **(415) 961-4103**

Letters *(Continued from page 34)*

Dear Editor:

To expand on Mr. Ham's comments ("Why Novices Use So Many Variables"): while I certainly agree that the stack effect of a word should not vary, here are a couple of reasons why novices (and others, too) might find it advantageous to refer to data as explicitly declared and named variables, rather than as carefully managed stack references.

1. Readability and maintainability of the program is improved, since the data is precisely identified as it is created and used, and any future changes in the algorithm that affect stack depth do not affect data references.

2. Except in very simple cases, management of the data on the stack (keeping track of what's where) consumes significant time and is prone to errors; therefore, programmer productivity is reduced.

Sincerely,

David Held
P.O. Box 483
Hermosa Beach, CA 90254

Mil-Std-1750, Anyone?

Gentlemen:

Fairchild produces the F9450 processor (bipolar, 18 MHz clock frequency) which executes the Mil-Std-1750 instruction set. Our question is whether you know of anybody who has implemented Forth targeting the Mil-Std-1750 instruction set. Or is there any chance that our request can be published in the next issue of *Forth Dimensions*?

Thank you very much for your help.

Sincerely,

Hans J. von der Pfordten
Systems Manager
Bipolar Microprocessor Products
Fairchild
450 National Ave.
Mountain View, CA 94043

DO...LOOP83 Caution

Dear Marlin:

The following may save people a lot of debugging time. More *hidden logic* has been added to **DO...LOOP83**. **DO83**'s run time **(DO)83** may not necessarily place the actual index and limit on the return stack. You may find instead that the number has been changed in some machine-dependent way.

The practical problem stems from *expectations* arising from **DO78** and **DO79** behavior. **(DO)78** and **(DO)79** place the actual limit and index on the stack in all the implementations I have seen (however, I have not seen all implementations). Such **(DO)** action is, to me, natural in the sense that what happens is directly related to the code one reads.

The '78 and '79 standards defined **R** and **R@** to return the value at the top of the return stack, whereas **I** is defined to return the value of the index. Forth-83 has the same definitions for **R@** and **I**.

I have seen a lot of code where **R** or **R@**, and **I** are used as if they are identical, which is technically incorrect but, in practice, the code works. In Forth-83, this "habit" from the past must be discarded and proper use of **R** or **R@** and **I** must be employed if you want your code to be transportable.

Sincerely yours,

Nicholas Pappas
1201 Bryant St.
Palo Alto, CA 94301

;CODE: vs. DOES>

Dear Editor:

;CODE: is the suggested name for a new word that performs the same function as **DOES>** but uses one word less storage in every word compiled by the defining word, in which it is used in the form

: ccc CREATE ... ;CODE: ...

At least as implemented in the 6502 fig-FORTH model, **<BUILDS... DOES>** uses the first word in the parameter field as a pointer to the high-level code following **DOES>** in the defining word, with the code field pointing to the routine which controls **IP** at run time. By contrast, **;CODE:** does not use the parameter field at all and uses the code field to point to the code immediately following **;CODE:** in the defining word. At compile time, **;CODE:** places a machine-language subroutine call in the defining word to a routine which controls **IP** with the same effect as **DOES>**. This works on installations using the machine return stack for the Forth return stack, and may be implemented as in figure one.

Figure One

```
HEX
: ;CODE:  COMPILE (;CODE)
jsr C, LIT [ HERE 2 ALLOT ] ,
  ; IMMEDIATE HERE SWAP !
  ASSEMBLER
```

...machine code to do the following:

1. Push W + 2 on the computation stack.
2. Swap IP and the top of the return stack.
3. Jump to NEXT

"jsr" is the literal value of the JSR opcode for the machine. Even when the machine and Forth return stacks are not the same, there is an equivalent implementation.

The word **;CODE:** is actually **DOES>** in 79-Standard. When writing nine months before publication of that standard, I used a different name to allow public reaction before changing a definition in the FIG Model (a practice to which Bill Ragsdale objected). At this time, 1983, many FIG Model users have still not converted their systems over to this new form, perhaps because the implementation was not clear. Instead of the code routine above, implementations like the FIG 8086 publication require the following routine because they use the machine stack for the parameter field rather than the return stack:

1. Push **IP** in the return stack.
2. Pop the parameter stack into **IP**.
3. Push **W+2** on the parameter stack (or just jump to the code for variables).

One can also use a JMP instead of a JSR, computing the implicit results of the JSR from the code field, which may actually be faster on some processors.

Sincerely yours,

George B. Lyons
280 Henderson St.
Jersey City, NJ 07302

```
SCR# 122
( PRICING PROGRAM - FORTH DEM DEC-1983 )
: UND*   )R OVER U* ROT R) * + ;  ( UN UD1 --- UD2 )
  DECIMAL
  2VARIABLE TOTAL
: .MONEY (# # # ASCII . HOLD #S ASCII $ HOLD #) TYPE SPACE ;
: +TOTAL TOTAL 2@ D+ TOTAL 2! ;
: PRICE CREATE , , DOES) 2@ UND* +TOTAL ;
: COST  CREATE , , DOES) 2@ +TOTAL ;
: LOCATES CREATE , DOES) @ LOAD ;
: DOZEN 12 * ;
: FIGURE 0 0 TOTAL 2! [COMPILE] FIND  EXECUTE TOTAL 2@ .MONEY ;
( DIRECTORY )
123 LOCATES PRICES
124 LOCATES FRUIT-BASKET
```

Manufacturing Elegance

Dear Editor,

In his "Manufacturing Cost Program" (*Forth Dimensions* V/4), Marc Perkel challenges "...anyone to write such an elegant program in any other language." Because I am at the novice level, one of the learning methods I use is to translate interesting programs from *Forth Dimensions* into MicroMotion Forth, and to try to make improvements if possible.

In **PRICE**, the words **>R OVER U* ROT R> * +** multiply an unsigned number by an unsigned double number and yield an unsigned, double-precision product. I defined these words as **UND* (UN UD1 --- UD2)** in the attached code. (Note: **0.** puts **0 0** on the stack and **FIND** replaces **'**.) Also, MicroMotion has a utility word **UDN* (UD1 UN --- UD2)** that is defined in assembly code, and a **ROT UDN*** works very nicely.

Yes, Perkel's program is elegant. But the real challenge is trying to maintain Forth's tradition of short, well-defined words, especially in a program being used by a manufacturer.

Sincerely,

Ronald E. Apra
Physics Department
Pioneer High School
1290 Blossom Hill Rd.
San Jose, CA 95118

When NOT is Not 0=

Dear FIG,

I'm grateful for anyone who warns me that the plate is hot before I burn my fingers. If any of you are about to convert an existing program to Forth-83, let me tell you how I burned my fingers, in hope of saving you the pain.

I had read all about the changes, and thought I was prepared for them. But two subtle changes crept up on me. The first is that **NOT** is now a one's complement operation. This is inconsistent with the historical use of **NOT** which was to reverse the effect of **IF**. For instance, we can write:

: TEST1 BLK @ IF ." Using Mass storage " **THEN ;**

i.e., if **BLK** contains non-zero, we're using mass storage. Traditionally, we could also define the reverse:

: TEST2 BLK @ NOT IF ." Using input message buffer " **;**

i.e., if the contents of **BLK** is *not* non-zero (it's zero), we're using the input message buffer.

The traditional **NOT** is identical with **0=**. The whole point of having two words with the same name is to enhance readability. With the 83-Standard, **NOT** no longer reverses the effect of **IF**. Assuming the contents of **BLK** is 80, then **NOT** returns -81, which **IF** will regard as true.

This problem appeared about five times in the program I just converted — even after I had thought I'd wrung out all the bugs. My solution in each case was to re-edit each **NOT** to a **0=**. In retrospect, I feel that the 83-Standard is simply wrong; in the future I will redefine:

: NOT 0= ;

If the need should arise for a one's complement operation, I'll use the traditional phrase **-1 XOR**. (If I ever felt the need to define this operation as a word, I'd name it more appropriately **INVERT**.)

The second unexpected problem arose with the new **LEAVE** which in the 83-Standard causes an immediate jump out of the loop. Now, I knew about that and carefully checked through the listing to make sure that every instance of **LEAVE** appeared at the end of the loop code, just before the **LOOP**.

But in verifying my conversion, I found that one loop sometimes left a value on the stack upon completion. This particular loop uses a variable increment each time through: if it **TYPE**s a long string, the length of the string is the increment; if it **EMIT**s a single character, one is the increment. After computing the appropriate increment, the loop ends with the phrase:

... DONE? IF LEAVE THEN +LOOP ;

I double-checked every word in the definition to see which one was leaving the extra value on the stack. Try as I might, I couldn't find it. Finally, I realized that, with **LEAVE** exiting the loop immediately, **+LOOP** was no longer consuming the final increment the last time through, as it had done before.

Having found the problem, the solution was simple:

... IF DROP LEAVE THEN +LOOP

But it seemed wrong that I should have to change the code in this way. I believe this is because the new **LEAVE** is now as dangerous as using **EXIT** in the middle of a definition; you must know what you are doing to make sure the stack comes out all right.

I hope these comments will serve to warn innocent users of these anomalies with the new standard system, and also to stimulate re-consideration by the Forth Standards Team of ideas that appear not to have been tested in actual implementations.

Sincerely,

Leo Brodie
17714 Kingsbury St.
Granada Hills, CA 91344

Products & Announcements

Competitions

The Forth Vendors Group has decided to take drastic action to help raise public consciousness about Forth. A "substantial" cash prize will be awarded to the author of the best article about Forth to be published in a general interest computer magazine during 1984. Rules, regulations and other fine print pertaining to this competition may be obtained by writing to: Ray Duncan, c/o Laboratory Microsystems Inc., P.O. Box 10430, Marina Del Rey, CA 90295.

Mountain View Press will award grants totalling $5000 to full-time college or university students, for the best entries of 2500 words or less describing "An EXPERT System Rule Set for Writing EXPERT System Rule Sets." First place award will be $2000, second place will be $1000 and twenty honorable mentions will receive $100 grants. Entries must be received by June 30, 1984 and postmarked by June 15, 1984. For complete rules, write to: Student Grant Competition, Mountain View Press Inc., P.O. Box 4656, Mountain View, CA 94040.

Classes

Humboldt State University presents two summer classes on Forth. June 18-21 are the dates of the introductory class; advanced material will be covered June 26-29. Tuition ranges from $125 - $200, depending on the course and if academic credit is desired. For information, call the Office of Continuing Education at 707-826-3731. Classes will be filled on a first-come, first-served basis.

New Products

MicroMotion has announced *Forth Tools*, a comprehensive text introducing the Forth-83 Standard and its extensions. Data structures, I/O and **CREATE...DOES>** are covered, and each chapter contains problems and solutions. The $20 book is the required text for UCLA and UC-Berkeley extension courses on Forth. Write to: MicroMotion, 12077 Wilshire #506, Los Angeles, CA 90025.

Henry Laxen and Michael Perry have implemented a public-domain Forth system conforming to the Forth-83 Standard. Only the following disk formats are available: 8" ss/sd CP/M-80 for 8080; 8" ss/sd CP/M-86 for 8086; 8" ss/sd CP/M-68K for 68000; and 5.25" ds/dd for IBM-PC MS-DOS. Each system costs $25, is public domain and comes with no visible support. Available from: No Visible Support Software, P.O. Box 1344, 2000 Center St., Berkeley, CA 94704.

q4th is available for CP/M 2.+ Z80 systems from Quanta Corp. It is a superset of Forth-79 and includes ROMable code, editor, assembler, debug, trig and other features. Introductory price is $95; outside of USA, add $30 shipping. 5" disk formats are Televideo, Epson, NorthStar, Zenith; and 8" ss/sd for IBM. Write: Quanta Corp., 2510 Sunset Blvd., Los Angeles, CA 90026.

PROA Corp. will make publicly available the control board developed for its line of automated machines. The multi-purpose controller is based on Rockwell's R65F11 chip, which contains Forth in ROM. The PROATROL can function as both development system and as dedicated controller. For price and on-board options, write: PROA Corp., 4019 Edith Blvd. NE, Building 2B, Albuquerque, NM 87107.

ACSG Inc. offers fig-FORTH for the Sage, adapted to the UCSD P-system 68000 assembler. Available on 5.25" diskettes for Sage II and IBM-PC; for $50. For information, write: ACSG, Inc., P.O. Box 40878, Tucson, AZ 85717.

CALL for PAPERS

1984 Asilomar Conference

Place: Pacific Grove, California
Sponsors: FORML (Forth Modification Laboratory)
Forth Interest Group
Time: November 23-25, 1984

Contact: Mr. Robert Reiling
Forth Interest Group
P.O. Box 1105
San Carlos, CA 94070

1984 Taipei FORML Conference

Place: Taipei, Taiwan
Republic of China
Sponsors: FIG Chapter,
Republic of China
FORML (Forth Modification Laboratory)
Forth Interest Group
Time: September 28-30, 1984
Contact: Dr. C. H. Ting
156 14th Ave.
San Mateo, CA 94402
(415) 571-7639

1984 Shanghai FORML Conference

Place: Shanghai,
People's Republic of China
Sponsors: Chiao-tung University, Shanghai
FORML (Forth Modification Laboratory)
Forth Interest Group
Time: October 3-5, 1984
Contact: Mr. Robert Reiling
Forth Interest Group
P.O. Box 1105
San Carlos, CA 94070

Chapter News

John D. Hall
Oakland, California

We have three new chapters — that makes forty-seven!

Richmond Forth Group
Richmond, Virginia

French Language FIG Chapter
Toulouse, France

Irish FIG Chapter
Waterford, Ireland

The purpose of this column is to report to you, the members, what each of the chapters is doing. I do not attend all the chapter meetings, so it is up to the chapters to send me a report of the happenings. Some of the newsletters are so good as they are, that I do not want to summarize them. Here, then, is the Kansas City FIG Chapter News, reprinted as presented:

Kansas City FIG Chapter News

Our last meeting was held November 22, 1983 with eighteen people attending. During the first part of the meeting, we went over the problems in chapters one and two of *Starting Forth*. The first part of the meeting has been dedicated to introducing Forth and helping beginners get started. Feedback is encouraged, so let's hear from you beginners. This is your hour! In the meantime, during subsequent meetings we will continue working the problems in *Starting Forth* until we are finished, and then we will start over. We have a computer and monitor available at the meetings and enough copies of the book such that everyone can follow along. Going over the problems together has been beneficial to everyone. Among other things, it has provided an impetus for discussing the various implementations and differences in versions of Forth that are available.

Also at our last meeting, Marty Sainsbury spoke on the topic "Programming Style and Organization." He directed a workshop on solving a telephone switching problem that Kim Harris presented at the 1981 Rochester Forth Conference. For more information, refer to the *1981 Rochester Forth Proceedings*. Donnal Walkers drew a Warnier-Orr diagram of the solution for the same problem. He uses this type of diagram to improve communication between programmers working on the same project.

Forth publications and programs offered by Mountain View Press are available at our chapter at a forty percent discount. We will finalize an order at our next meeting, December 27, 1983. Please take advantage of this offer and have your order ready. If you cannot attend the next meeting, please send your order to me immediately.

We will receive newsletters and handouts from other FIG chapters when they are made available to the Chapter Coordinator, John Hall in California. What we have received will be available for review at our meetings and can be checked out. This newsletter will be sent to John Hall for distribution to other FIG chapters after approval of the newsletter is made at our next meeting. If you wish to add or change something, please let me know before or during our next meeting.

Notes of appreciation:
Bill Jellison for getting our meeting hall.
Bill Pitts for copies of the newsletter.

Kansas City FIG Chapter News
January 17, 1984

Our last meeting was held December 27, 1983. Few people were able to attend, so our agenda for the last meeting has been re-scheduled for the next meeting, January 23, 1984. We will do problems and exercises in *Starting Forth*. Our topic for discussion is "Metacompilers." Les Lovesee will give a demonstration of his metacompiler.

We finally have an order to send to Mountain View Press. I believe we can get another large order in six months. Keep this in mind. We have close to fifty people in our chapter and about half attend the meetings regularly. We can purchase anything offered by MVP or FIG at a forty percent discount for orders greater than $1,000. If there is anything you wish to order, let me know and we will start getting a new order together.

Our schedule for meetings in 1984 is as follows:

Place: Midwest Research Institute
Mag Conference Center
Time: 7:00 - 9:00 p.m.
Dates:
Mon., Jan. 23
Tues., Feb. 28
Mon., Mar. 26
Tues., Apr. 24
Mon., May 28
Tues., June 26
Tues., July 24
Tues., Aug. 28
Mon., Sept. 24
Tues., Oct. 23
Tues., Nov. 27
(Dec. to be scheduled)

Contact: Linus Orth, work: 816/444-6655

FIG of NYC

At the November meeting, Redmond Simon demonstrated his enhancements to the GraFORTH Apple II graphics package.

At the December meeting, the group began what will become an ongoing group project, attempting to formulate and implement a virtual terminal interface standard, which would allow the creation of transportable editors. Also in December, the FIG of NYC was granted tax-exempt status by the Internal Revenue Service as a publicly supported organization.

Missing Cities?

As I searched across the map of FIG chapters, I noticed that several cities were missing. *MISSING???* In these cities, there are sufficient FIG members, there are many new people interested in Forth, and there is the need for a chapter. I can only ask, why don't we have chapters in these cities? Is it a lack of missionaries, a shortage of heroes, a deficit (thanks, trusty thesaurus) of expertise? No, sadly, it is probably from a malaise brought on by the effects of an insufficient amount of sleep by our FIG members in their pursuit of the higher goal of spreading the word of Forth.

For any of you that do not qualify above, figure one provides a list of forty-three cities where there are more than enough FIG members to get a chapter organized. Just send a note asking for a Chapter Kit to:

John Hall
National Chapter Coordinator
P.O. Box 1105
San Carlos, California 94070

Newark, NJ	Colorado Springs, CO
Pittsburg, PA	Lompoc, CA
Tallahassee, FL	Arcata, CA
Columbus, OH	Spokane, WA
South Bend, OH	Ithaca, NY
Kalamazoo, MI	Durham, NC
Milwaukee, WI	Miami, FL
Omaha, NE	Cincinnati, OH
Baton Rouge, LA	Lafayette, IN
Lubbock, TX	Madison, WI
Reno, NV	Urbana, IL
Monterey, CA	New Orleans, LA
Eugene, OR	San Antonio, TX
Schenectady, NY	Salt Lake City, UT
Blacksburg, VA	Ridgecrest, CA
Gainesville, FL	Corvallis, OR
Toledo, OH	Kodiak, AK
Ft. Wayne, IN	Tokyo, Japan
Ames, IA	Helsinki, Finland
Rochester, MN	Copenhagen, Denmark
Lincoln, NE	Amsterdam, Netherlands
Oklahoma City, OK	

Figure One

Chapters in Formation

Here are more of the new chapters that are forming. If you live in any of these areas, contact these people and offer your support and help in forming a FIG chapter. You are not expected to be one of the "Forth experts." The job of organizing a chapter may well be better left to the people who are better in organizing than in programming, or to people who are in need of the help and support that a chapter can return. Lend a hand!

RENEW NOW!

SEND A CHECK TO FIG TODAY!

Tom Konantz
7808 Logan Dr.
Huntsville, AL 35802
205/881-6483

John C. Mead
3325 E. Tera Alta Blvd.
Tucson, AZ 85716

Chuck Larrieu
P.O. Box 294
Corte Madera, CA 94925
415/457-8791

Tom Ghormley
1315 E St.
Sacramento, CA 95814

Charles A. Krajewski
205 Blue Rd.
Middletown, CT 06457

Charles B. Duff
KRIYA
505 North Lake Shore Dr.
Chicago, IL 60611
312/822-0624

Michael J. Hannah
Sandia National Lab
Organization 2614
Albuquerque, NM 87185
505/846-3459

William L. Edmonds
1716 Bailey Road
Grafton, VA 23692

B. Lambey
151 rue Pierre Cardinal
36100 Montpellier
France

Ken McAllister
P.O. Box 8100
Christchurch
New Zealand

FORTH INTEREST GROUP

MAIL ORDER

	USA	FOREIGN AIR
☐ Membership in FORTH Interest Group and Volume V of FORTH DIMENSIONS	$15	$27
☐ Back Volumes of FORTH DIMENSIONS. **Price per each.** ☐I ☐II ☐III ☐IV	$15	$18
☐ fig-FORTH Installation Manual, containing the language model of fig-FORTH, a complete glossary, memory map and installation instructions	$15	$18
☐ Assembly Language Source Listings of fig-FORTH for specific CPU's and machines. The above manual is required for installation. Check appropriate box(es). **Price per each.** ☐1802 ☐6502 ☐6800 ☐6809 ☐VAX ☐Z80 ☐8080 ☐8086/8088 ☐9900 ☐APPLE II ☐ECLIPSE ☐PACE ☐NOVA ☐PDP-11 ☐68000 ☐ALPHA MICRO	$15	$18
☐ "Starting FORTH, by Brodie. BEST book on FORTH. (Paperback)	$18	$22
☐ "Starting FORTH" by Brodie. (Hard Cover)	$23	$28
☐ PROCEEDINGS: FORML (FORTH Modification Conference) ☐ 1980, $25USA/$35Foreign ☐ 1981, Two Vol., $40USA/$55Foreign ☐ 1982, $25USA/$35Foreign ROCHESTER FORTH Conference ☐ 1981, $25USA/$35Foreign ☐ 1982, $25USA/$35Foreign ☐ 1983, $25USA/$35Foreign	Total	$_____
☐ STANDARD: ☐ FORTH-79, ☐ FORTH-83. $15USA/$18Foreign EACH.	Total	$_____
☐ Kitt Peak Primer, by Stevens. An in-depth self-study book.	$25	$35
☐ MAGAZINES ABOUT FORTH: ☐ BYTE Reprints 8/80-4/81 ☐ Dr Dobb's Jrnl, ☐ 9/81, ☐ 9/82, ☐ 9/83 ☐ Poplar Computing, 9/83 $3.50USA/$5Foreign EACH.	Total	$_____
☐ FIG T-shirts: ☐ Small ☐ Medium ☐ Large ☐ X-Large	$10	$12
☐ Poster, BYTE Cover 8/80, 16"x22"	$ 3	$ 5
☐ FORTH Programmer's Reference Card. If ordered separately, send a stamped, self addressed envelope.		Free
	TOTAL	$_____

NAME_____MS/APT_____

ORGANIZATION_____PHONE( )_____

ADDRESS_____

CITY_____STATE_____ZIP_____COUNTRY_____

VISA#_____MASTERCARD#_____

AMERICAN EXPRESS#_____Card Expiration Date_____

(Minimum of $15.00 on Charge Cards)

Make check or money order in US Funds on US Bank, payable to: FIG. All prices include postage. No purchase orders without check. California residents add sales tax. 10/83

ORDER PHONE NUMBER: (415) 962-8653

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

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

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

Address Correction Requested