THE BEST OF $1.00

FORTH DIMENSIONS

MEETINGS

PUZZLES

LITERATURE

MATH

INTRODUCTION

CONFERENCES

STYLE

MARKETING

LANGUAGE

CARTOONS

STANDARDS

LETTERS

GRAPHICS

LEARN FORTH
JOIN FIG!

The best way to learn FORTH and keep up with implementation and application information is to join the FORTH Interest Group. You will receive each issue (six) of FORTH Dimensions as it is published and you will be able to join a local FIG Chapter.

Membership costs $15.00 US, or $27.00 Foreign and runs concurrent with the magazine year. Volume V covers from May, 1983 through April, 1984. Back Volumes I, II, III, and IV are available for $15.00, US or $18.00 Foreign.

☐ **Yes,** I want to join FIG and receive all of Volume V.

Name ______________________________

Organization ______________________________

Address ______________________________

City ______________________________

State ______________________ ZIP __________

VISA ______________________________

MASTERCARD ______________________________

Expiration Date for charge card: ______________________________

Make check or money order in US Funds on US Bank, payable to **FIG.** All prices include postage. California residents add sales tax except on current membership. No purchase orders accepted without checks.

ORDER PHONE NUMBER: 415/962-8653

FORTH INTEREST GROUP   PO Box 1105   San Carlos, Calif. 94070

FORTH DIMENSIONS

JUNE/JULY 1978 VOLUME 1 NO. 1

EDITORIAL:
WHAT IS THE FORTH INTEREST GROUP?

The Forth Interest Group, which developed in the fertile ground of the computer clubs of the San Francisco Bay Area, grew in a few months from nothing to where we are now getting several letters a day from all over the country. With this increasing public interest we need to let people know what we are doing and why, what we would like to see happen, how others can be involved, and what we can and cannot do.

We are involved because we believe that this language can have a major effect on the usefulness of computers, especially small computers, and we want to see it put to the test. Increasingly software is becoming the critical, limiting factor in the computer industry. Large software projects are especially difficult to develop and modify. Few are happy with prevailing operating systems, which are huge, hard to understand, incompatible with each other, and without unity of design.

The Forth language is its own operating system and text editor. It is interactive, extensible. (including user-defined data types), structured, and recursive. Code is so compact that the entire system (mostly written in Forth) usually fits in 6K bytes, running stand-alone with no other software required, or as a task in a conventional operating system. One person can understand the entire Forth system, change any part of it, or even write a new version from scratch. Run-time efficiencies are as little as 30% slower than straight machine code, and even less if the system's built-in assembler is used. When the assembler is not used, programs can be almost completely transportable between machines. Any large Forth program is really a special-purpose, application-oriented language, greatly facilitating maintenance and modification. We don't yet have conclusive data, but typical program development times and costs seem to be a fraction of those required by Fortran or assembly. Forth is especially useful for real-time, control-type applications, for large projects, and for small machines.

The problem is availability. Users have shown an ease of learning after they have a system available. The Forth characteristics of postfix notation, structured conditionals, and data stacks are best understood by use. To encourage Forth programmers, we need readily available systems even of modest performance. We hope that three levels will be available:

1. Demonstration - free (or under $20.) introductory version without file structure which compiles and executes from keyboard input.

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

2. Personal - low cost ($10. to $100.) with RAM or tape based files.

3. Professional - Commercial products for lab or industrial use and software development. ($1000. to $2500.)

Today the serious personal computer user holds the key to wider availability of the language. These users - generally engineers, businessmen, programmers - combine professional competence and commitment with the freedom to try new methods which may require a lot of time and tinkering with no definite guarantee of payoff. Practically everyone involved with the Forth Interest Group has both a personal and a professional interest in computers.

The Forth Interest Group is non-profit and non-commercial. We aren't associated with any vendor, no one is making money from it, and we are all busy with other work. We are an information clearinghouse and want to encourage distribution of all three of the previously mentioned levels of Forth. We do not have a Forth system for distribution at this time, and we don't want to get into the software or mail-order business because this is best left to companies or individuals committed to that goal. Naturally our critical issue is how to keep going over the long haul with volunteer energy. We need cost-effective means of information exchange.

At present we are writing for professional media, putting out this simple newsletter, and holding occasional meetings in the Bay Area. Also, we are developing a major technical and implementation manual, to be published in a journal form as four installments, available by subscription. While we cannot answer all of the mail individually, we certainly read it all, to answer it in the newsletter. While we cannot fill orders for software or literature, we will try and point you to where it is available. We welcome your input of information or suggestions, how you could help, what you would like to see happen, and where we should go from here.

Dave Bengel - Dave Boulton - Kim Harris - John James
Tom Olsen - Bill Ragsdale - Dave Wyland

TOWERS OF HANOI

by Peter Midnight

Here are the listings of a graphic representation of the ancient Towers of Hanoi puzzle which is adjustable for any CRT terminal with curser addressing.

Recently, when I got fig FORTH running on my system under North Star DOS, I decided to translate this program into FORTH as an exercise and as a comparison between FORTH and PASCAL. In the process I noticed some inefficiencies but chose to translate them more or less directly, for the sake of comparison.

The UCSP PASCAL program is available by requesting the Jan/Feb 1980 Newsletter from Homebrew Computer Club, P.O. Box 626, Mountain View, CA 94042.

Forth Program

```
SCR # 12
  0 ( TOWERS OF HANOI      Copyright, 1979, Peter Midnight )
  1 ( Translated for speed comparison ) FORTH DEFINITIONS DECIMAL
  2 ( First extend Forth to include a few features of Pascal )
  3 : MYSELF  ( In definition, this is a recursive use of new
  4    LATEST PFA CFA , ; IMMEDIATE                           word )
  5 : GOTOXY   ( X Y GOTOXY )   27 EMIT  61 EMIT
  6   0 MAX  15 MIN  32 +  EMIT   0 MAX  63 MIN  32 +  EMIT  ;
  7 : CLEARSCREEN  12 EMIT ;
  8 : 2DROP  DROP DROP ;
  9 : PICK  SP@  SWAP  2 *  +  @  ;
 10 : 4DUP  4 PICK  4 PICK  4 PICK  4 PICK ;
 11    10 CONSTANT NMAX  ( maximum permisable number of rings )
 12 NMAX VARIABLE (N)   : N (N) @ ; ( formerly a constant )
 13    0 CONSTANT HELL_FREEZES_OVER   43 CONSTANT COLOR ( + )
 14    0 VARIABLE RING   N 2 -  ALLOT  ( array [1..N] of bytes )
 15   -->

SCR # 13
  0 ( TOWERS OF HANOI      Copyright, 1979, Peter Midnight )
  1 : DELAY   ( centiseconds DELAY )
  2   0 DO  17 0 DO  127 127 * DROP  LOOP  LOOP  ;
  3 : POS     ( location POS -> coordinate )
  4   2 N *  1+  *  N +  ;
  5 : HALFDISPLAY   ( color size HALFDISPLAY )
  6   0 DO   DUP EMIT   LOOP   DROP  ;
  7 : <DISPLAY>   ( line color size <DISPLAY> )
  8   2DUP HALFDISPLAY   ROT 3 <  IF BL  ELSE 124 ( | )
  9   THEN EMIT   HALFDISPLAY  ;
 10 : DISPLAY   ( size pos line color DISPLAY )
 11   SWAP >R ROT ROT OVER - R ( color size pos-size line )
 12   GOTOXY   R> ( color size line )  ROT ROT <DISPLAY> ;
 13 -->
 14
 15

SCR # 14
  0 ( TOWERS OF HANOI      Copyright, 1979, Peter Midnight )
  1 : PRESENCE   ( tower ring PPESENCE -> boolean )
  2   RING +  C@  =  ;
  3 : LINE        ( tower LINE -> display_line_of_top )
  4   4 SWAP  N 0 DO  DUP I PRESENCE 0=  ROT + SWAP  LOOP DROP ;
  5 : 1-   1 - ;
  6
  7 : RAISE   ( size tower RAISE )
  8   DUP   POS  SWAP  LINE   1  SWAP  DO
  9   2DUP I BL DISPLAY   2DUP  I 1-  COLOR DISPLAY
 10   -1 +LOOP   2DROP  ;
 11 : LOWER   ( size tower LOWER )
 12   DUP   POS  SWAP  LINE 1+  2 DO
 13   2DUP  I 1-  BL DISPLAY   2DUP I COLOR DISPLAY
 14   LOOP  2DROP  ;
 15 -->

MSG # 15

SCR # 15
  0 ( TOWERS OF HANOI      Copyright, 1979, Peter Midnight )
  1 : MOVELEFT  ( size source_tower destiny_tower MOVELEFT )
  2   POS 1-  SWAP POS 1- DO  DUP  R 1+  1  BL  DISPLAY
  3   DUP R 1 COLOR DISPLAY   -1 +LOOP   DROP  ;
  4 : MOVERIGHT ( size source_tower destiny_tower MOVERIGHT )
  5   POS 1+  SWAP POS 1+ DO  DUP  R 1-  1  BL  DISPLAY
  6   DUP R 1 COLOR DISPLAY      LOOP    DROP  ;
  7 : TRAVERSE  ( size source_tower destiny_tower TRAVERSE )
  8   2DUP >   IF MOVELEFT  ELSE MOVERIGHT   THEN ;
  9 : MOVE   ( size source_tower destiny_tower MOVE )
 10       ?TERMINAL IF  0 N 4 + GOTOXY  ABORT  THEN
 11   ROT ROT 2DUP RAISE   >R 2DUP R> ROT TRAVERSE
 12   2DUP  RING + 1-  C!    SWAP LOWER  ;
 13 -->
 14
 15

SCR # 16
  0 ( TOWERS OF HANOI      Copyright, 1979, Peter Midnight )
  1 : MULTIMOV  ( size source destiny spare MULTIMOV )
  2   4 PICK  1 =  IF DROP MOVE      ELSE
  3   >R >R SWAP 1- SWAP R> R>       4DUP SWAP MYSELF
  4   4DUP DROP   ROT 1+ ROT ROT   MOVE
  5   ROT ROT SWAP   MYSELF    THEN  ;
  6
  7 : MAKETOWER   ( tower MAKETOWER )
  8   POS  4 N +  3 DO  DUP I GOTOXY  124 EMIT ( | )  LOOP DROP ;
  9 : MAKEBASE   ( no arguments )
 10   0 N 4 + GOTOXY  N 6 * 3 +  0 DO  45 EMIT ( - )  LOOP ;
 11 : MAKERING   ( tower size MAKERING )
 12   2DUP RING + 1- C!   SWAP LOWER ;
 13 : SETUP   ( no arguments )   CLEARSCREEN
 14   N 1+ 0 DO  1 RING I + C!  LOOP   3 0 DO  I MAKETOWER  LOOP
 15   MAKEBASE   0 N DO  0 I MAKERING  -1 +LOOP  ;   -->

SCR # 17
  0 ( TOWERS OF HANOI      Copyright, 1979, Peter Midnight )
  1 : TOWERS   ( quantity TOWERS )
  2   1 MAX  NMAX MIN  (N) !
  3   SETUP   N 2 0 1    BEGIN
  4   OVER POS  N 4 +  GOTOXY   N 0 DO  7 EMIT  50 DELAY  LOOP
  5   ROT   4DUP   MULTIMOV
  6   HELL_FREEZES_OVER UNTIL  ;
  7
  8 ;S
  9
 10 ( Results:  DELAY runs much slower in Forth than in Pascal.
 11   But the rest of the program is over twice as fast in Forth!
 12
 13   Note that CLEARSCREEN and GOTOXY are terminal dependant.
 14   NMAX should be 10 for 16x64 or 12 for 24x80 screens. )
 15

MSG # 15
```

Thanks to "THE I/O PORT", the Official Newsletter of the Tulsa Computer Society, for the feature article on FORTH by Art Sorski in their April 1980 issue. Address: The Tulsa Computer Society, P.O. Box 1133, Tulsa, OK 74101.

D-CHARTS

Kim Harris

An alternative style of flowcharts called D-charts will be described. But first the purpose of flowcharting will be discussed as well as the shortcomings of traditional flowcharting.

A flowchart should be a tool for the design and analysis of sequential procedures which make the control flow of a procedure clear. With FORTH and other modern languages, flowcharts should be optimized for the top-down design of structured programs and should help the understanding and debugging of existing ones. An analogy may be made with a road map. This graphic representation of data makes it easy to choose an optimum route to some destination, but when driving, a sequential list of instructions is easier to use (e.g., turn right on 3rd street, left on Ave. F, go 3 blocks, etc.). Indentation of source statements to show control structures is helpful and is recommended, but a two dimensional graphic display of those control structures can be superior. A good flowchart notation should be easy to learn, convenient to use (e.g., good legibility with free-hand drawn charts), compact (minimizing off-page lines), adaptable to specialized notations, language, and personal style, and modifiable with minimum redrawing of unchanged sections.

Traditional flowcharting using ANSI standard symbols has been so unsuccessful at meeting these goals that "flowchart" has become a dirty word. This style is not structured, is at a lower level than any higher level language (e.g., no loop symbol), requires the use of symbol templates for legibility, and forces program statements to be crammed inside these symbols like captions in a cartoon.

D-charts have a simplicity and power similar to FORTH. They are the invention of Prof. Edsger W. Dijkstra, a champion of top-down design, structured programming, and clear, concise notation. They form a context-free language. D-charts are denser than ANSI flowcharts usually allowing twice as much program to be displayed per page. There are only two symbols in the basic language; however, like FORTH, extensions may be added for convenience.

Sequential statements are written in free form, one below the other, and without boxes.

```
statement
next statement
next statement
   .
   .
   .
```

The only "lines" in D-charts are used to show nonsequential control paths (e.g., conditional branches, loops). In a proper D-chart, no lines go up; all lines either go down or sideways. Any need for lines directed up can be (and should be) met with the loop symbols. This simplifies the reading of a D-chart since it always starts at the top of a page and ends at the bottom.

It is customary to underline the entry name (or FORTH definition name) at the top of a D-chart.

2-WAY BRANCH SYMBOL

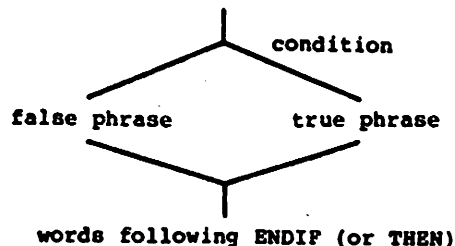
In FORTH, this structure takes the form:

```
condition IF   true phrase
          ELSE  false phrase
          THEN  .
```

Another FORTH structure which is used for conditional compilation has more mnemonic names:

```
condition IFTRUE   true phrase
          OTHERWISE   false phrase
          ENDIF   .
```

The D-chart symbol has parts for each of these elements:

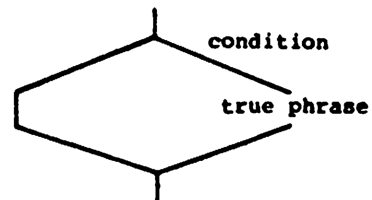


The "condition" is evaluated. If it is true, the "true phrase" is executed; otherwise, the "false phrase" is executed. The words following ENDIF (or THEN) are unconditionally executed.

If either phrase is omitted, as with

condition IF   true phrase   THEN

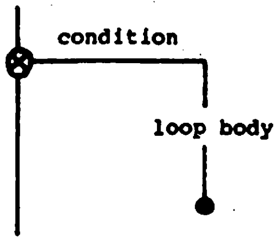
a vertical line is drawn as shown:



-->

LOOP SYMBOL

The basic loop defining symbol for D-charts is properly structured.



The switch symbol:

indicates that when the switch is encountered, the "condition" (on the side line) is evaluated.

1. If the "condition" is true, then the side line path is taken; if false, then the down line is taken (and the loop is terminated).

2. If the side line is taken, all statements down to the dot are executed. The dot is the loop end symbol and indicates that control is returned to the switch.

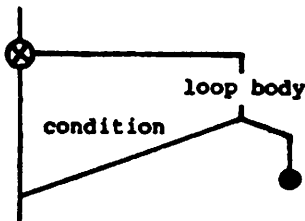
3. The "condition" is again evaluated. Its outcome might have changed during the execution of the loop statement.

   Repeat these steps starting with Step 1.

This symbol tests the loop condition before executing the loop body. However, other loops test the condition at the end of the loop body (e.g., DO .. LOOP and BEGIN .. END) or in the middle of the loop body. This loop symbol may be extended for these other cases by adding a test within the loop body. Consider the FORTH loop structure

```
BEGIN   loop body   condition END   .
```

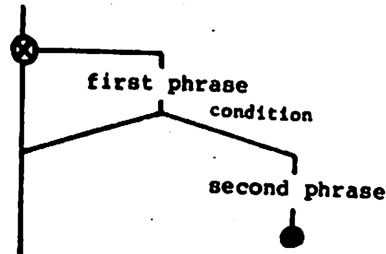
The loop body is always executed once, and is repeated as long as condition is false. The D-chart symbol for this structure would be:



A more general case is

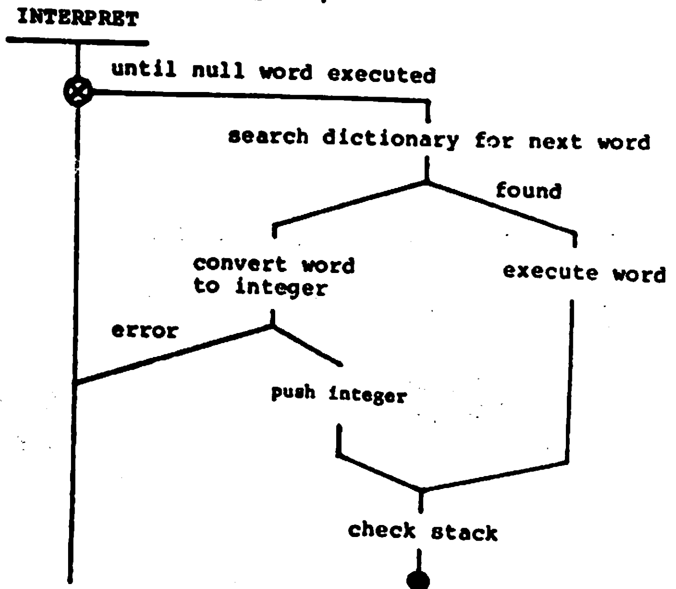
```
          BEGIN   first phrase
condition IF   second phrase
          AGAIN
```

which is explained better graphically than verbally:



Both previous symbols may be properly nested indefinitely. The following example shows how these symbols may be combined. This is the FORTH interpreter from the F.I.G. model.

```
: INTERPRET   BEGIN   (')   IF HERE NUMBER
                            ELSE EXECUTE
                            THEN
                      ?STACK
              AGAIN   ;
```

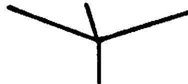


-->

n-WAY BRANCH SYMBOL

A structured n-way branch symbol (sometimes called a CASE statement) may be defined for convenience. (It is functionally equivalent to n nested 2-way branches). One style for this symbol is:

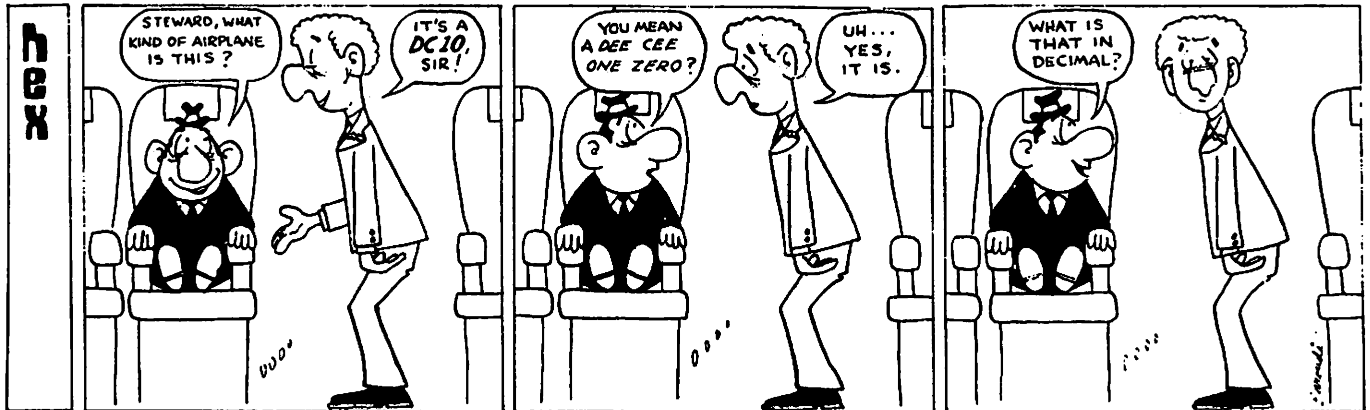
first case   second case   ..   last case



The condition is usually an index which selects one of the cases. The rejoining of control to a single line after the cases are required by structured programming. Depending on the complexity of the cases, this symbol may be drawn differently.

D-charts are efficient and useful. They are vastly superior to traditional flowchart style.

;S KIM HARRIS



Techniques Tutorial

Choosing Names

Henry Laxen

This time I would like to rant and rave about one of the most difficult aspects of programming in FORTH, that of choosing good names for your definitions. Besides a rational design, this is the single most important part of programming in FORTH.

That's a strong statement, but it is absolutely true. The names you give your definitions can make the difference between understandable, modifiable code, and complete garbage. I will illustrate this by some examples and some guidelines of how to choose good names.

First a word on programming tools. There has been a great deal of time and effort devoted to the topic of programming tools in recent years, and FORTH is well equipped with some of the most sophisticated tools in the software world. You can find code for countless debuggers, decompilers, cross reference utilities, glossary generators, and online helpers of one form or another. These are all wonderful, but rarely is the most important FORTH development tool mentioned, yet it is widely available and costs only about $20.00. I am of course talking about a good dictionary and thesaurus. When it comes to choosing a good name for a FORTH word, these can be invaluable, and should be part of every FORTH programmer's tool kit.

Now then, rule number one in choosing good names is: *Name the what, not the how.* Let's take a look at some examples of what this means. Every FORTH programmer, myself included, is guilty of violating this rule, and the primary violation is in the area of returning booleans or truth values. Every piece of code I have ever seen has phrases such as the following:

```
IF DO-SOMETHING 1 ELSE DROP 0 THEN
```

This is horrible! Furthermore there is a proposal in the 83 Standard to change the value of a true boolean from 1 to −1. If that happens, many many programs will need to be heavily modified. What we have in essence done in the above example is violate our rule on naming clarity. We have named the words TRUE and FALSE with the how, namely **1** and **0**, instead of the what, namely **TRUE** and **FALSE**. A much better solution, and it is absolutely trivial to implement, is to revise the code as follows:

```
1 CONSTANT TRUE
0 CONSTANT FALSE
IF  DO-SOMETHING TRUE ELSE
  DROP FALSE THEN
```

This is great! First it is absolutely clear that we are returning a boolean value, and secondly if this was done throughout, changing the value of **TRUE** would be little more than redefining the constant **TRUE**. The result is clearer, more understandable, and more modifiable code than before. [Editor's note: Or consider using **T** and **F** as abbreviations. See the code for my QTF article on page 21 of this issue.]

Let's look at another example of naming the what and not the how. It is often desirable to define some words which will set a variable to 1 or 0, **TRUE** or **FALSE**. Which of the following pieces of code have you written, and which do you now think is better:

```
: 0! ( addr -- ) 0 SWAP ! ;
: 1! ( addr -- ) 1 SWAP ! ;

: SET ( addr -- ) TRUE SWAP ! ;
: RESET ( addr -- ) FALSE SWAP ! ;
```

Suppose we had a variable called **ENABLE**. Which of the following phrases do you think makes more sense:

ENABLE 0! or **ENABLE RESET**

If you ask yourself *how* am I going to disable something, you will come up with the **0!** name. If you ask yourself *what* am I going to do, the answer will be to **RESET** the **ENABLE** flag, and you will come up with the much superior name of **RESET** instead of **0!**.

Always remember to ask yourself what you are doing, not how you are doing it. If you answer the what question, you will most likely come up with a good name.

Now let's proceed to rule number two in how to choose a good name. Rule 2 is: *If possible, stick to English.* Given the choice between good, ordinary, prosaic English, and super sophisticated computerese, *always* choose English.

Besides a rational design, choosing good names is the most important part of programming in FORTH.

Let's take a look at an example of this rule. What name would you give to the word that takes a row and column position off the stack and moves the cursor of your terminal to that position? Think about it for a minute before you read the next paragraph.

If you chose a word like **GOTOXY** or **XYPOS** may you burn in the fires of PASCAL forever! These are total computerese gibberish, and should be avoided like the plague. A terrific word for this function would be **AT**, since you are positioning the cursor **AT** the values that are on the stack. (This name was stolen by me from Kim Harris who credits Chuck Moore.) Compare how much more nicely the code fragment:

5 20 AT ." Hello" reads compared to **5 20 GOTOXY ." Hello"**

Let's take another example, which might be sacrosanct to many of you. Suppose you wanted to define a word which will list all of the words in a particular vocabulary on your terminal. What would be a good name for such a beast? If you said **VLIST** try again. **VLIST** is another example of computerese gibberish. If you would like to know what the **EDITOR WORDS** are doesn't it make more sense to type **EDITOR WORDS** than **EDITOR VLIST**? **WORDS** is the perfect name for such a function. It names the what, namely

Continued on page 35

tell me what the **WORDS** are, not the how of Vocabulary LISTing.

Now let's take a look at the third and final rule in choosing a good name. Rule number 3 is: *All things being equal between two names, choose the shorter one.* Let's try our rules on the following problems. Think of a name for a word that will clear the screen of a video terminal. Some names that immediately spring to mind are: **ERASE**, **BLANK** and **CLEAR**. Unfortunately **ERASE** and **BLANK** are already taken, and **CLEAR** seems like a good choice, but maybe we can do better. **CLEAR** could apply to other things besides a video terminal, so think about words that would only apply to visual things. Consider the word **DARK**. This is ideal for this function. All things being equal between **CLEAR** and **DARK** we would choose **DARK** based on rule 3. Let's look at one more example. What name should I give the word that decompiles other FORTH words. The syntax I want is

??? **NAME**

where ??? will decompile the FORTH word **NAME**. Think of what we are doing and come up with some names. Rule 2 excludes garbage such as **DECOMP** and **DIS**. What is it we are doing? We are exposing the definition of **NAME**. Think of words that mean expose. How about the following: **EXPOSE DISCLOSE REVEAL SEE**. They are all good English words that describe what is going on.

For a long time I used **REVEAL** for this function, but then later I finally came up with **SEE**, and chose it based on rule number 3. I don't see any intrinsic value of **SEE** over **REVEAL** other than it is shorter, and hence easier to type. Both **SEE QUIT** and **REVEAL QUIT** appeal to me.

As a final example, and perhaps a piece of useful code that you can use in your applications, let's take a look at Fig. 1. This example was motivated by a frequent occurrence in many of my programs, namely that of returning a TRUE or FALSE result during some kind of searching procedure. Furthermore, this returned result must be capable of nesting properly.

For example, suppose we wanted to search a string for an occurrence of a control character. If you are passed the address and length of the string, you might wind up with a piece of code as shown in Fig. 2.

It first shoves a **0** behind the address and length on the stack. It then runs through the string character by character and if it finds a control character, it throws away the current address and the **0**, replaces them with two **1**s, and leaves the loop. After the loop, the address is thrown away, leaving only the boolean result. I think this is not only hard to follow, but tricky, and should be avoided.

Now compare it with the piece of code in Fig. 3. It starts out by saying that the result to be returned is initially false. Next it also runs through the string character by character, and if it finds a control character it simply indicates that the search was successful and leaves. After the loop the address is thrown away and the result is returned. What could be simpler and more readable?

Now let's examine Fig. 1 in more detail. The word **INITIALLY** is nothing more than a push onto a stack pointed to by the word **BOOLEANS**. Similarly **RESULT** is nothing more than a pop from the same stack. Notice how completely different the names are from how they are actually implemented. If I had named the how instead of the what, I would have wound up with names like **>BOOL** and **BOOL>**. Not only would this violate rule number 1, but it would be complete computer gibberish as well. How many of you have implemented stacks with names such as **>GARBAGE** and **GARBAGE>**?

Just because something is a stack doesn't mean it has to have little arrows associated with it. Stacks are very useful data structures, and when you use them to implement a function, be sure to name them according to what the function does, not how it does it. I have found that using the code in Fig. 1 has improved the readability of my programming immensely, at almost zero cost.

In conclusion, I would like to leave you with the immortal words of the poet John Keats, from his poem ENDYMION. He said something like: "A good name is a joy forever." Till next time, may the FORTH be with you. □

©Henry Laxen 1982

Henry Laxen is an independent FORTH consultant based in Berkeley, California.

```
Scr # 1
  0 \ Fig 1.       Boolean Results                                11SEP82HHL
  1 CREATE BOOLEANS   0 ,   20 ALLOT   ( Space for the stack )
  2 : INITIALLY    ( n -- )
  3     BOOLEANS   2 OVER +!    ( increment index )
  4     DUP @ + !    ( and store n )   ;
  5 : RESULT       ( n -- )
  6     BOOLEANS   DUP   DUP @ + @ ( get top of stack )
  7     -2 ROT +! ( and decrement index )    ;
  8 : FAIL         ( -- )
  9     RESULT DROP   FALSE INITIALLY    ;
 10 : SUCCEED      ( -- )
 11     RESULT DROP    TRUE INITIALLY    ;
 12
 13
 14
 15

Scr # 2
  0 \ Fig. 2.   Poor way to Search a String                      11SEP82HHL
  1 : CONTROL?    ( addr len -- f )
  2     0 ROT ROT    0 DO    DUP C@ BL
  3         < IF    2DROP   1 1    LEAVE    THEN
  4     LOOP    DROP    ;
  5
  6
  7 \ Fig. 3.   Neat way to Search a String
  8 : CONTROL?    ( addr len -- f )
  9     FALSE INITIALLY    0 DO    DUP C@ BL
 10         < IF    SUCCEED    LEAVE    THEN
 11     LOOP    DROP    RESULT    ;
 12
 13
 14
 15
```

FORTH in Literature

At the FORTH Convention, October, 1979, Dan Slater gave a short report on an experiment on communication with killer whales. By use of a touch sensitive plate, the orca could learn to physically equate touching a position with a concept or object. Interest was expressed in using the syntax of FORTH to define new items. By this method a man/whale vocabulary can be built.

The evening Charles Moore read a FORTH poem by Ned Conklin. It is loosely based on a classic of English literature.

```
:    SONG
       SIXPENCE  !
  BEGIN  RYE  @  POCKET  +!  ?FULL  END
  24  0  DO  BLACKBIRD  I  +  @  PIE  +!  LOOP
  BAKE  BEGIN  ?OPENED  END
  SING  DAINTY-DISH  KING  !  SURPRISE  ;
```

A-21

Bill Ragsdale has submitted two more. This is a familiar quotation, with apologies to Browning:

```
:    LOVE
     CR ." How do I love thee?"
     CR ." Let me count the ways."
         1 BEGIN   CR   DUP   .   1+ AGAIN   ;
```

```
:    RHYME
       JACK  DUP  NIMBLE  BE
             DUP  QUICK  BE
       CANDLE-STICK  OVER  JUMP  ;
```

Finally here is an actual, full poem. It is taken from "The Space Childs Mother Goose" by Frederick Winsor, Simon and Schuster, 1958. It consists of eleven stanzas and is almost recursive.

The first two screens compile the primitives from which the poem is recited, by loading of the last screen. The computer's recitation occurs stanza by stanza with the operator indicating his interest and approval by operating any terminal key at the REST after each stanza.

```
SCR # 108
  0 (  The Theory that Jack built                          WFR-79DEC15 )
  1 ( From The Space Child's Mother Goose, Frederick Winsor )
  2 : RECITE       110  LOAD  QUIT  ;   ( say this poem )
  3 : THE                   ." the "    ;
  4 : THAT         CR       ." That "    ;
  5 : THIS         CR       ." This is "  THE   ;
  6 : JACK                  ." Jack built"  ;
  7 : SUMMARY               ." Summary"  ;
  8 : FLAW                  ." Flaw"  ;
  9 : MUMMERY               ." Mummery"  ;
 10 : K                     ." Constant K"  ;
 11 : HAZE                  ." Erudite Verbal Haze"  ;
 12 : PHRASE                ." Turn of a Plausible Phrase"  ;
 13 : BLUFF                 ." Chaotic Confusion and Bluff"  ;
 14 : STUFF                 ." Cybernetics and Stuff"  ;
 15 : THEORY                ." Theory "  JACK  ; -->
```

```
SCR # 109
  0 (  More Poem                                           WFR-79DEC15 )
  1 : BUTTON            ." Button to Start the Machine"  ;
  2 : CHILD             ." Space Child with Brow Serene"  ;
  3 : CYBERNETICS     ." Cybernetics and Stuff"  ;
  4 : HIDING        CR  ." Hiding " THE  FLAW  ;
  5 : LAY        THAT  ." lay in "  THE  THEORY  ;
  6 : BASED      CR    ." Based on "  THE  MUMMERY  ;
  7 : SAVED         THAT  ." saved "  THE  SUMMARY  ;
  8 : CLOAK  CR    ." Cloaking "  K  ;
  9 : THICK IF THAT ELSE CR ." And " THEN ." Thickened " THE HAZE ;
 10 : HUNG          THAT  ." hung on " THE  PHRASE    ;
 11 : COVER IF THAT ." covered " ELSE CR ." To cover " THEN BLUFF ;
 12 : MAKE       CR    ." To make with "  THE  CYBERNETICS  ;
 13 : PUSHED     CR    ." Who pushed "  BUTTON  ;
 14 : REST       46  EMIT  10  SPACES  KEY  DROP  CR  CR  CR  ;
 15 : WITHOUT  CR ." Without Confusion, exposing the Bluff" ; RECITE
```

```
SCR # 110
  0 (  Recite our poem                                     WFR-79DEC15 )
  1 CR  CR  CR  THIS  THEORY  REST
  2 THIS  FLAW  LAY  REST
  3 THIS  MUMMERY  HIDING  LAY  REST
  4 THIS  SUMMARY  BASED  HIDING  LAY  REST
  5 THIS  K  SAVED  BASED  HIDING  LAY  REST
  6 THIS  HAZE  CLOAK  SAVED  BASED  HIDING  LAY  REST
  7 THIS PHRASE 1 THICK CLOAK SAVED BASED HIDING LAY REST
  8 THIS BLUFF HUNG 1 THICK CLOAK SAVED BASED HIDING LAY REST
  9 THIS STUFF 1 COVER HUNG 0 THICK CLOAK SAVED BASED HIDING
 10   LAY REST
 11 THIS BUTTON  MAKE 0 COVER  HUNG 0 THICK  CLOAK  SAVED
 12   BASED  HIDING  LAY  REST
 13 THIS CHILD PUSHED CR ." That made with " CYBERNETICS  WITHOUT
 14   HUNG CR ." And, shredding " THE HAZE CLOAK CR ." Wrecked " THE
 15   SUMMARY BASED HIDING CR  ." And Demolished " THEORY REST
```

MARKETING COLUMN

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

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

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

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

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

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

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

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

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

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

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

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

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

STRUCTURED PROGRAMMING BY ADDING MODULES TO FORTH

Dewey Val Schorre

Structured programming is a strong point of FORTH, yet there is one language feature important for structured programming which is currently absent in FORTH. This feature is called a module in the programming language MODULA, and appears under other names in other languages, such as procedure in PASCAL. It can, however, be easily added by defining three one-line routines.

The names of these routines are: INTERNAL, EXTERNAL and MODULE. A module is a portion of a program between the words INTERNAL and MODULE. Definitions of constants, variables and routines which are local to the module are written between the words INTERNAL and EXTERNAL. Definitions which are to be used outside the module are written between the words EXTERNAL and MODULE.

One of the most common uses of modules is to create local variables for a routine. These variables are defined between INTERNAL and EXTERNAL. The routine which references them is defined between EXTERNAL and MODULE. Notice that this module feature is more general than the local variable feature of other programming languages, in that several routines can share local variables. Such sharing is important, not so much from the standpoint of saving space, but because it provides a means of communication between the routines.

If you have written any local routines between the words INTERNAL and EXTERNAL, then in order to debug them, you will have to delete the word INTERNAL and put a ;S before the word EXTERNAL. Since debugging in FORTH proceeds from the bottom up, once you have debugged these local routines, you will have no further need to refer to them from the console. They will only be referenced from the external part of the module. Modules can be nested to arbitrary depth. In other words, one module can be made local with respect to another by defining it between the words INTERNAL and EXTERNAL.

Now let's consider matters of style. The matching words INTERNAL, EXTERNAL and MODULE should all appear on the same screen. When modules are to be nested, one should not actually write the lower level module between the words INTERNAL and EXTERNAL, but should write a LOAD command that refers to the screen containing the lower level module. The screens of a FORTH program should be organized in a tree structure. The starting screen which you LOAD to compile the program is a module which LOAD's the next level modules.

Screens are much better for structured programming than the conventional character string file because they can be chained together in this tree structured manner. You will write a module for one program, and when you want to use it in another program, you don't have to edit it into the new program or add it to a library. All you have to do is to reference it with a LOAD command.

There is an efficiency advantage to the use of modules. One minor advantage is that compilation speed is improved because the dictionary that has to be searched is shorter. The more important advantage of saving dictionary space is not realized with this simple implementation, which changes a link in the dictionary. To save space, one would have to implement a dictionary that

was separate from the compiled code. Moreover, this dictionary would not be a simple push-down stack, because the storage freed by the word MODULE is not the last information entered into the dictionary.

The words needed to define modules are as follows:

```
: INTERNAL  ( --> ADDR)  CURRENT @ @ ;
: EXTERNAL  ( --> ADDR)  HERE ;
: MODULE( ADDR1 ADDR2 --> )PFA LFA ! ;
```

FORML CONFERENCE

A Report on the
Second FORML Conference

The Second Conference of the Forth Modification Laboratory (FORML) was held over Thanksgiving, November 26 to 28, 1980, at the Asilomar Conference Center, Pacific Grove, California (some 120 miles south of San Francisco).

The weather was unseasonably beautiful, as the rainy season, normally starting in November, was late. Most conference attendees managed to find some free time to enjoy the beach and wooded areas.

With the way smoothed by a core crew who showed up Tuesday, the majority of participants arrived for lunch Wednesday, and launched right into a full schedule of technical sessions.

There were 65 conference attendees, with enough of them bringing family to raise the count to 96 people at Asilomar in connection with FORML.

The rooms were in scattered well-landscaped buildings. Meals were provided in a central dining building, and were generally praised. Thanksgiving noon dinner, a deluxe buffet meal, was a special treat.

The evening meetings, both Wednesday and Thursday, had formal technical sessions which evolved into quite open, informal, and productive discussions. The participants had to be persuaded to break up to move to the scheduled social gatherings over wine and cheese.

SUMMARY OF SESSIONS

The number of people presenting papers was so great (almost 40) that sessions were scheduled from Wednesday afternoon all the way to Friday afternoon. Topics of sessions, together with their chairmen, were:

FORTH-79 Standard
Bill Ragsdale

Implementation Generalities
Don Colburn

Implementation Specifics
Dave Boulton

Concurrency
Terry Holmes

FORTH Language Topics
George Lyons

Other Languages
Jon Spencer

MetaFORTH
Armand Gamberra

Programming Methodology
Eric Welch

Applications
Hans Niewenhuijzen

In addition, Kim Harris, the Conference Chairman, opened the Conference with a welcome and a review of FORML-1, London, January 1980. Kim also closed the final session.

LETTERS TO THE EDITORS

Dear Fig:

I have developed a process-simulation program that occupies very little memory space and yet has many of the capabilities of commercial simulation packages.

I have been heavily involved in modeling and simulation of automated manufacturing systems for over six years. My ultimate objective for this work is to develop a microprocessor-based simulation capability which incorporates process control structures far beyond those of currently available languages. However, the relatively extensive modeling power of the current code would seem to offer interesting market potential in its own right.

If you can provide information on marketing such a product, please contact me by mail or by phone (home (317) 447-9206, office (317) 749-2946).

Joseph Talavage, Ph.D.
3907 Prange Dr.
Lafayette, IN 47905

Hope printing your letter helps.--ed.

Dear Fig:

I am puzzled as to why I have not seen mention in your New Products announcements of fullFORTH+ for PET, available also, I believe for Apple. It is published by IDPC, Co., PO Box 11594, Bethlehem Pike, Colmar, PA 18915 at $65. It is advertised as "A full-featured FORTH with extensions conforming to Forth Interest Group standards. Includes assembler, string processing capabilities, disk virtual memory, multiple dimensioned arrays, floating point and integer processing." Surely, fullFORTH+ is worth a mention, if not a comprehensive review!

Francis T. Chambers
ROCK HOUSE
Ballyoroy, Westport
Co. Mayo, Ireland

Thank you for your interest. This was reviewed in Vol. III, #3.--ed.

Dear Fig:

This is our response to Chuck's (Moore's) cute letter.

Arthur Goldberg
Spencer SooHoo
CEDARS SINAI MEDICAL CTR.
9700 Beverly Blvd.
Los Angeles, CA 90048

DEA- CHU--

WE CHA------ YOU TO CON----- THE CON----- OF THI- CON------ LET---. WE CLA-- THA- THE CON---- OF A WOR- IN COM--- ENG---- CON-------- CON-------- TO OUR ABI---- TO DEC----- IT FRO- THR-- LET---- AND THE LEN---.

HOW----, IN COM----- PRO----- THE CON---- IS LES- CON------ (COM----, WOR-- CAN BE SWA---- WIT---- CHA----- THE SEM------). CON--------- IT CON-------- CON--------- LES- HEL- IN IDE-------- THE DEF------- OF A WOR-. IN FAC-, AS THI- LET--- DEM---------, THR-- LET---- AND A LEN--- CAN LET---- EVE- A CAR---- REA--- OF COM--- ENG----.

SIN------ YOU--,
ART--- GOL-----
SPE---- SOO---

(TRANSLATION)

Dear Chuck:

WE CHALLENGE YOU TO CONSIDER THE CONTENTS OF THIS CONFUSING LETTER. WE CLAIM THAT THE CONTEXT OF A WORD IN COMMON ENGLISH CONTRIBUTES CONSIDERABLY TO OUR ABILITY TO DECIPHER IT FROM THREE LETTERS AND THE LENGTH.

HOWEVER, IN COMPUTER PROGRAMS THE CONTEXT IS LESS CONFINING (COMMONLY, WORDS CAN BE SWAPPED WITHOUT CHANGING THE SEMANTICS). CONSEQUENTLY IT CONTRIBUTES CONSIDERABLY LESS HELP IN IDENTIFYING THE DEFINITION OF A WORD. IN FACT, AS THIS LETTER DEMONSTRATES, THREE LETTERS AND A LENGTH CAN LETDOWN EVEN A CAREFUL READER OF COMMON ENGLISH.

SINCERELY YOURS,
ARTHUR GOLDBERG
SPENCER SOOHOO

Your letter and its "translation" certainly make the point!--ed.

Dear Fig:

Right now I am trying to put together a local Danish FIG, and I would therefore like you to update me with the names and addresses of the Danish FIG members and possible make a note in FORTH DIMENSIONS about my intentions.

As the communication lines are rather long and since our magazine is only bimonthly, please inform me on your next deadline as soon as possible.

Niels Oesten
Brostykkeveh 189
DK-2650 Hvidovre
Denmark

Thanks Niels. Good luck on establishing a local Danish FIG Group. Anyone interested, please contact Niels as listed above. Regarding deadlines: Copy must be in our hands 6 weeks prior to publication, i.e., 4/15 is the deadline for May/June edition, etc.--ed.

Dear Fig:

I just wanted to write to tell you how much I enjoy FORTH DIMENSIONS. Every issue has several things of interest to me, and I appreciate your work in seeing that it gets done (often a thankless task). Here in New Hampshire, Rob Moore of SNAC (Southern New-Hampshire Apple Corps) is doing most of the work in implementing and refining a version of fig-FORTH for the Apple II. We have taken as much as possible from page zero so that we can use the many subroutines available from the Applesoft ROM. I have been working with our version for some time now and am doing a high-resolution graphics game using FORTH and Applesoft hi-res routines.

Gregg Williams
BYTE Publications
PO Box 372
Hancock, NH 03449

Thanks Gregg. Glad you enjoy and appreciate our efforts.--ed.

Dear Fig:

Regarding the 8080 Renovation Project's requests for bug fixes, I would like to counter with a request that they provide a status report in FORTH DIMENSIONS that includes those bugs already reported along with any solutions proposed or implemented. It would also be of interest to find out what the goals are for the 8080 Renovation Project and how local FIG chapters can help.

There is what I consider a bug in that the message routine uses an absolute value of screen 4 and 5 for getting error message information. This is fine where offset is zero but when an offset other than zero is used and the disk has other information on absolute screens 4 and 5, things don't look too good.

Robert I. Demrow
P. O. Box 158 BluSta
Andover, MA 01810

Thanks for the input.

LEARN FORTH

JOIN FIG!

The best way to learn FORTH and keep up with implementation and application information is to join the FORTH Interest Group. You will receive each issue (six) of FORTH Dimensions as it is published and you will be able to join a local FIG Chapter.

Membership costs $15.00 US, or $27.00 Foreign and runs concurrent with the magazine year. Volume V covers from May, 1983 through April, 1984. Back Volumes I, II, III, and IV are available for $15.00, US or $18.00 Foreign.

☐ **Yes,** I want to join FIG and receive all of Volume V.

Name ______________________________

Organization ______________________________

Address ______________________________

City ______________________________

State ______________________________ ZIP __________

VISA ______________________________

MASTERCARD ______________________________

Expiration Date for charge card: ______________________________

Make check or money order in US Funds on US Bank, payable to **FIG.** All prices include postage. California residents add sales tax except on current membership. No purchase orders accepted without checks.

ORDER PHONE NUMBER: 415/962-8653

FORTH INTEREST GROUP PO Box 1105 San Carlos, Calif. 94070

Floating Point FORTH?

By MICHAEL JESCH
Aregon Systems, Inc.

One of the first things most programmers find missing in FORTH is floating point arithmetic. While most implementers of FORTH probably weigh the advantages and adversities of floating point for their version, they usually decide to forego it for various reasons. On the other hand, some excellent floating point systems have been developed in FORTH. This article overviews some major problems with floating point numbers, and examines a very rudimentary floating point system, written entirely in high level FORTH.

The first major problem with floating point numbers is that no computer can work with numbers that are truly floating point. Instead, quite often they are stored as two separate numbers, mantissa and exponent. This leads to

The greatest advantage of a floating point math system comes in ease of use

the second major problem, that of speed. Because each floating point number is stored as two separate numbers, each function requires that two numbers be dealt with, costing quite a bit in speed. One of the most common solutions for speed problems is to buy a dedicated processor chip to do all the arithmetic. This is impossible on some computers, and costly on others.

Another problem, that of accuracy, is a prime consideration. While floating point numbers can have a greater range, their precision suffers a little. The system outlined in this article has precision to only six full decimal digits, but the decimal point can be moved 127 places in either direction. This compared to a normal 32 bit double length number, where the range and accuracy are +/-2,147,483,647.

The greatest advantage of a floating point math system comes in ease of use: there is no need for the programmer to worry about where the decimal point should be, as it is handled internally by the floating point operators themselves. This saves time programming and debugging, and usually saves some memory too.

Continued on next page

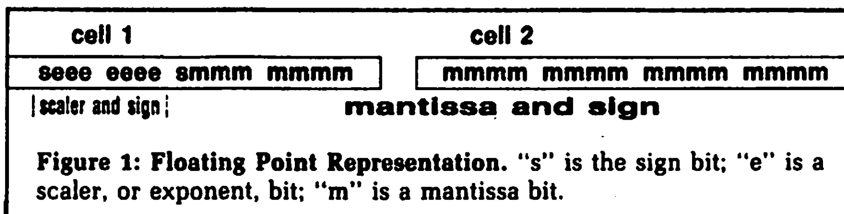


Figure 1: Floating Point Representation. "s" is the sign bit; "e" is a scaler, or exponent, bit; "m" is a mantissa bit.

```
1000  LIST

  0  ( Floating point       MCJ    11/02/81 )
  1
  2  VARIABLE  FPSW           ( floating point status word )
  3
  4  FPSW 1+ CONSTANT FBASE       ( base )
  5
  6  : FRESET    ( -- )       ( clear condition codes )
  7       0 FPSW C! ;
  8
  9  : FINIT     ( -- )       ( initialize processor )
 10       FRESET
 11       BASE @    FBASE  ! ;
 12
 13
 14
 15


1001  LIST

  0  ( Floating point       MCJ    11/02/81 )
  1
  2  : FER     ( -- n )    ( returns sum of condition codes )
  3       FPSW C@ ;
  4
  5  : FZE     ( -- n )    ( true if last F# was zero )
  6       FER  1 AND  0= NOT ;
  7
  8  : FNE     ( -- n )    ( true if last F# was < zero )
  9       FER  2 AND  0= NOT ;
 10
 11  : FOV     ( -- n )    ( true if last operation was overflow )
 12       FER  4 AND  0= NOT ;
 13
 14  ( CC's 8, 16, 32, 64, and 128 are available )
 15


1002  LIST

  0  ( Floating point       MCJ    11/02/81 )
  1  HEX
  2
  3  : SFZ     ( F# -- F# ; Z )    ( sets Z according to F# )
  4       FER  FFFE AND  FPSW C!      ( reset Z )
  5       2DUP  00FF AND  D0=    FER OR    FPSW C! ;
  6
  7  : SFN     ( F# -- F# ; N )    ( sets N according to F# )
  8       FER  FFFD AND  FPSW C!      ( reset N )
  9       DUP  0080 AND  40 /  FER OR  FPSW C! ;
 10
 11  DECIMAL
 12
 13
 14
 15
```

Listing continued on next page

This floating point system is written in high level FORTH as an educational tool. Once the machine language of the target machine is understood, it should be rewritten in low level to capitalize on speed. One important side effect/advantage of this approach is transportability: It has since been implemented on two other computers, under different FORTH systems (polyFORTH and MMSFORTH); one with a different CPU (an LSI-11/23). This approach does, however, cost a lot of execution time.

As can be seen in Figure One, the floating point number is represented in two 16-bit cells on the stack. The high cell (cell 1) contains the 8-bit exponent (containing one sign bit) and the high 8 bits of the mantissa, including the mantissa sign, while the low cell (cell 2) contains the lower 16 bits of the mantissa. In this manner, the existing double length stack and memory operators can be used to manipulate the values.

The error detection is handled by the system, but the recovery is left up to the programmer. A special condition code 'register' is used to return information about the last operation. Currently, three of these bits are used: One each to indicate the occurrence of a zero value, a negative value, and most overflow/underflow conditions. These flags can be tested, as in a FORTH **IF** . . . **THEN** structure, with words defined in the package (**FZE FNE** and **FOV**). The word **FER** will return a true if any error existed.

The scaler is used to tell where the decimal point is, relative to the ones' column of the mantissa, during all math operations and output. A positive value indicates that the radix point is actually to the right of the ones' column and by how many digits, while a negative value means to move it to the left. It could be considered a 'times **BASE** to the **SCALER**' type of suffix to a number. For addition and subtraction, the scalers must be made equal (the word **ALIGN** does this). This means shifting the mantissa the number of places equal to the difference of the exponents, which

```
1003 LIST
 0  ( Floating point     MCJ   11/02/81 )
 1  HEX
 2
 3  : @EXPONENT   ( F# -- M E ; Z N )   ( remove exponent )
 4       FRESET  SFZ  SFN               ( set flags )
 5       DUP  FF00 AND  100 /  >R       ( obtain exponent )
 6       FNE IF                         ( sign extend mantissa )
 7         FF00 OR
 8       ELSE
 9         00FF AND
10       THEN  R> ;
11
12  DECIMAL
13
14
15

1004 LIST
 0  ( Floating point     MCJ   11/02/81 )
 1  HEX
 2
 3  : 'EXPONENT   ( M E -- F# ; V Z N )   ( restores exponent )
 4       DUP  100 *  DUP 100 /  ROT <> IF
 5         4  FPSW C!                       ( exponent overflow )
 6       THEN
 7       SWAP DUP  FF00 AND DUP IF
 8         DUP  FF00 <> IF
 9           4 FPSW C!                 ( mantissa overflow )
10         THEN
11       THEN DROP
12       00FF AND   OR
13       SFZ SFN ;
14
15  DECIMAL

1005 LIST
 0  ( Floating point     MCJ   11/02/81 )
 1  : F.    ( F# -- ; Z N )
 2       @EXPONENT >R
 3       SWAP OVER  DABS
 4       <#   I 0< IF
 5         I ABS  0  DO   #   LOOP       46 HOLD
 6       ELSE
 7         46 HOLD
 8         I IF
 9           I  0  DO   48 HOLD   LOOP
10         THEN
11       THEN   R> DROP
12       #S   SIGN #>  TYPE SPACE ;
13
14  : E.    ( F# -- ; Z N )
15      @EXPONENT  <ROT  (D.) TYPE  ." . E" . ;

1006 LIST
 0  ( Floating point     MCJ   11/02/81 )
 1
 2  : F*     ( F#1 F#2 -- F# ; N Z V )   ( multiply )
 3       2SWAP  @EXPONENT >R
 4       2SWAP  @EXPONENT >R
 5       DROP  1  M*/
 6       R>  R> +   'EXPONENT ;
 7
 8
 9  : F/     ( F#1 F#2 -- F# ; N Z V )   ( multiply )
10       2SWAP  @EXPONENT >R
11       2SWAP  @EXPONENT >R
12       DROP  1 SWAP  M*/
13       R>  R> +   'EXPONENT ;
14
15
```

```
1007  LIST
 0  ( Floating point     MCJ   11/02/81 )
 1
 2  : ALIGN   ( M1 E1  M2 E2 -- M1 M2 E )
 3      BEGIN
 4        >R  ROT >R
 5      I'  I <> WHILE
 6        I'  I > IF     ( I' is E2 )
 7          2SWAP   FBASE C@  1  M*/    2SWAP    R> 1+ <ROT  R>
 8        ELSE
 9          R> <ROT   FBASE C@   1 M*/   R> 1+
10        THEN
11      PEPEAT
12      R>   R> DROP ;
13
14
15

1008  LIST
 0  ( Floating point     MCJ   11/02/81 )
 1
 2  : F+   ( F#1  F#2  -- FSUM  ; N V Z )
 3      2SWAP  @EXPONENT >R
 4      2SWAP  R> <ROT  @EXPONENT
 5      ALIGN   >R
 6      D+    R> !EXPONENT ;
 7
 8  : F-   ( F#1  F#2  -- FDIFF  ; N V Z )
 9      2SWAP  @EXPONENT >R
10      2SWAP  R> <ROT  @EXPONENT
11      ALIGN   >R
12      D-    R> !EXPONENT ;
13
14
15

1009  LIST
 0  ( Floating point     MCJ   11/02/81 )
 1
 2  : RSCALE    ( F# -- F# ; N Z V )
 3      @EXPONENT 1+  <ROT
 4      FBASE C@  1 M*/  ROT
 5      !EXPONENT ;
 6
 7  : LSCALE    ( F# -- F# ; N Z V )
 8      @EXPONENT 1-  <ROT
 9      1  FBASE C@ M*/  ROT
10      !EXPONENT ;
11
12
13
14
15

1010  LIST
 0  ( Floating point     MCJ   11/02/81 )
 1  : FIX    ( F# -- D# ; V Z N )
 2      @EXPONENT
 3      BEGIN
 4      ?DUP  WHILE
 5        DUP 0< IF
 6          1+  <ROT    FBASE C@   1  M*/
 7        ELSE
 8          1-  <ROT    1  FBASE C@  M*/
 9          2DUP D0= IF
10            5 FPSW C!   ROT DROP 0 <ROT    ( underflow )
11          THEN
12        THEN
13        ROT
14      REPEAT ;
15
```

Listing continued on next page

causes most of the imprecision problems present in this system. Furthermore, if one number was entered in hexadecimal (base 16) and the other in decimal, the scalers would be incompatible. To help circumvent this, a floating point base value is kept separate from the FORTH base value, and all internal scaling oprations use this value for the number base. The floating point base is set to the current FORTH base when the floating point system is initialized (with **FINIT**). It can also be explicitly set by the programmer, but be careful with this; if you output a number in a different base than you did arithmetic, the results will not be correct.

Number formatting is left up to the programmer, as it is in most FORTH systems. A double length number may be converted to floating point by inserting the desired scaler (number scaler **!EXPONENT**). To change a number from floating point to integer, the word **FIX** will scale the mantissa to zero. The scaler of the top floating point number can be extracted with the word **@EXPONENT**, which leaves the double precision mantissa below, unmodified. **F.** and **E.** are used to output the top floating point number. **F.** prints in floating point format (i.e., 123.45), while **E.** prints in scientific notation (i.e., 12345 E –2).

The four basic arithmetic functions, add, subtract, multiply and divide, are called **F+**, **F–**, **F*** and **F/**, respectively. **RSCALE** and **LSCALE** are used to change the position of the least significant digit in the mantissa. **RSCALE** decrements the scaler and multiplies the mantissa by base (changes 12.3 to 12.30), while **LSCALE** increments the scaler and divides the mantissa by base (changes 12.34 to 12.3). Be careful with these words, however. If the number is close to the limit of precision, the number will probably lose accuracy.

Other miscellaneous words are **FABS**, **FNEGATE**, **FMIN**, **F>**, and **FMAX**; these are the floating point counterparts to **ABS**, **NEGATE**, **MIN**, **>**, and **MAX**, respectively.

GRAPHIC GRAPHICS

Bob Gotsch
California College of Arts and Crafts

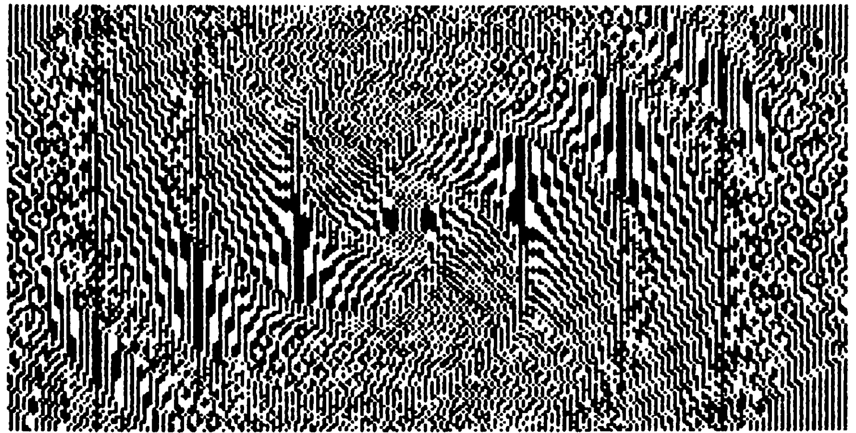
Accompanying these comments are several graphic specimens drawn on Apple computer using FORTH and printed on a dot-matrix printer. They range from logotype design to experiments in geometry and pattern. One can generate real-time motion graphics on the Apple in which color and action partially compensate for the low resolution of 280 by 192 pixels. Hardcopy, whether prinout or color photo, isn't the final product. The interactive, sequenced and timed display on the screen is the designed product, likely to displace the medium of print on paper in the future.

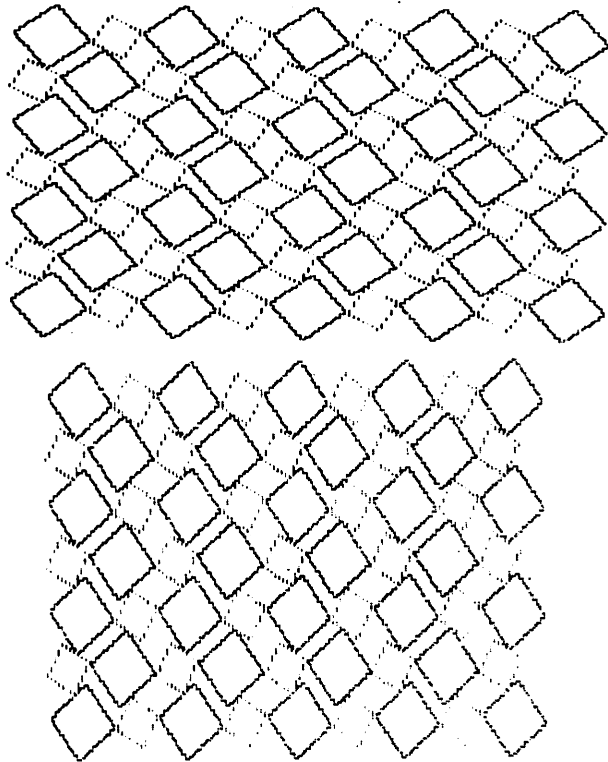
While these graphic samples could have been programmed in other languages, I have found the advantages of using FORTH are both practical and expressive: immediate and modular experimentation with the peculiarities and limitations of the Apple video display, and orchestration of complex visual effects with self-named procedures rather than the tedious plots and pokes to undistinguished addresses. With this ease of wielding visual ideas, FORTH might lead to a new era of computer graphics, even creative expression.

It may remain individual and personal expression, however, without graphics standards. Transportability of grahics--generating code may be neither possible nor desirable considering the differences in video display generation, alternate character sets, shape tables, display lists, interrupts, available colors, etc., between microcomputers. Each has some individual features to exploit. Most have, however, such limited memory for graphics as to make machine-dependent economy an overriding aspect of programming for graphics.

Despite the rarity of FORTH graphics thus far, I'm convinced it is an excellent vehicle for bringing out undiscovered graphics potential of each micro. In addition, the visibility gained by some effort to evolve grahic ideas in FORTH would help in both spreading and teaching the language. Perhaps this issue of FORTH DIMENSIONS will stimulate just such activity.

Editor's Note: The author tells me that Osborne/McGraw-Hill publishers have used his patterns, generated on Apple II using Cap'n Software FORTH, as cover artwork for their book "Some Common BASIC Programs"!





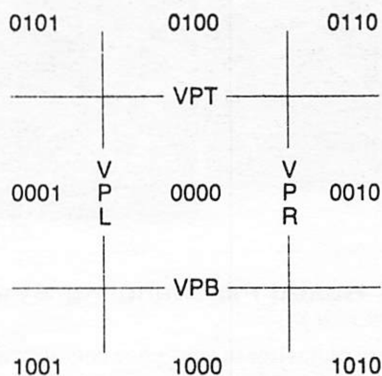
FORTH

The Sheer Joy of Clipping Recursively

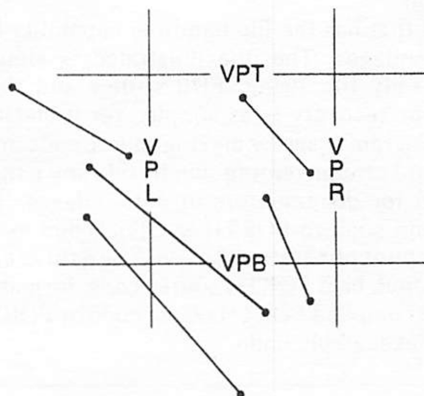
Bob Gotsch

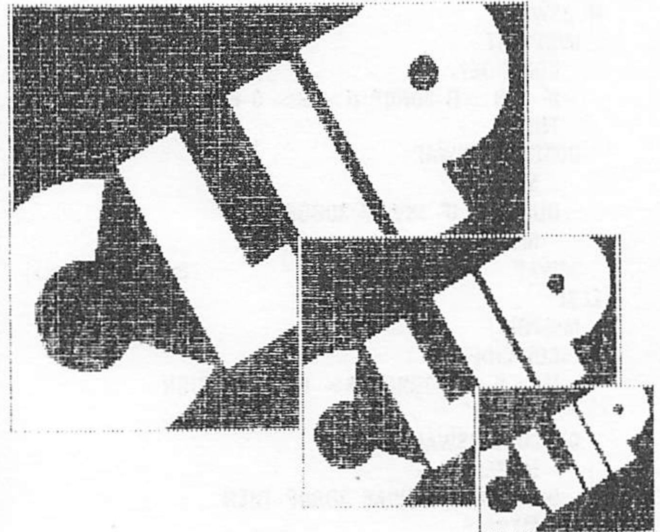
I've been wondering since I wrote the last article whether recursion would be useful for anything else but a study of recursion. That other thing turned out to be "clipping," discarding those parts of the lines of a picture that lie outside the specified "viewport," so that plotting takes place only within the desired or useable portion of the display device. One strategy for clipping is to perform a binary search for the visible extremes of each line, then plot the segment between.

Using the Sutherland-Cohen algorithm, 4-digit "outcodes" are assigned to the endpoints of a line according to where they lie, inside or outside the viewport.



Inside is 0000. For an X-value to the left of (less than) the viewport boundary, the rightmost bit is set (0001). Below the viewport the leftmost bit is set (1001). The outcode for lower left is the logical OR of left and below (1001), etc. A line can be trivially rejected (no plotting at all) if it lies entirely to one side (outside) of the viewport. An efficient test for rejection, returning a true flag, is the logical AND of the endpoint's outcodes. A line may be trivially accepted and plotted as-is when outcodes for both ends are zero.





Of all the possible locations of a line, entirely inside viewport, entirely outside, one end in, or crossing viewport, every situation is handled by trivial rejection, trivial acceptance, or successive middle divisions of the line until each of the segments of the line can be trivially accepted or trivially rejected. The binary search for the intersection with viewport boundary terminates when the segment becomes so short that the midpoint in integer screen coordinates coincides with one or the other endpoint.

I have chosen to save the three values for each endpoint, X-value, Y-value, and outcode together on the stack, with the out-of-viewport point always topmost on the stack. Hence a true **INVIEWPORT?** condition is followed by **3SWAP**. The other tests **TRIVIALACCEPT?**, **TRIVIALREJECT?**, and **COINCIDE?** as well as calculation of **MIDPOINT**, assignment of **OUTCODE**, and the graphics action **PLOTLINE** do just what they say and should be understandable from the foregoing without listed definitions.

These are incorporated in the recursive procedure **REJECT?** which uses the last outcode as true or false flag. If true, it returns to toplevel **CLIPLINE** having dropped one of the points, or at any lower level having droped off the half of the current line that is entirely outside of the viewport. Between **MYSELF**s is a test that drops the midpoint if it happened to be inside the viewport, so the search can continue onward for the other visible extreme of the line, conducted by the other **MYSELF**. If the actions of a recursive procedure are planned in advance, then the procedure-as-a-whole can be written to follow those rules, and each recursive call can be trusted to follow those rules. If **REJECT?** returns false, then **CLIPLINE** must plot the segment represented by the two endpoints left on the stack.

Continued

```
: RETURN R> DROP ;
: REJECT? ( 2x,2y,2oc,1x,1y,1oc
                                    --- 2x,2y,TF)
                     ( --- 2xcl,2ycl,2oc,1xcl,1ycl,FF)
  TRIVIALREJECT? IF 3DROP RETURN THEN
  TRIVIALACCEPT? IF RETURN THEN
    INVIEWPORT?
    IF 3SWAP                      (swap endpoints)
      MIDPOINT
        COINCIDE?
        IF >R >R 3DROP R> R> 0 RETURN
        THEN
      OUTCODE 3SWAP
          MYSELF
        DUP NOT IF 3SWAP 3DROP THEN
          MYSELF
      3SWAP                       (swap ends back)
    ELSE
      MIDPOINT
        COINCIDE?
        IF >R >R 3DROP R> R> 0 RETURN
        THEN
      OUTCODE 3SWAP
          MYSELF
        DUP NOT IF 3SWAP 3DROP THEN
          MYSELF
    THEN ;
: CLIPLINE ( 1x,1y,2x,2y --- )
   OUTCODE 5 ROLL 5 ROLL OUTCODE
     REJECT? DUP
     IF DROP 2DROP
     ELSE STRIPOUTCODES PLOTLINE
     THEN ;
```

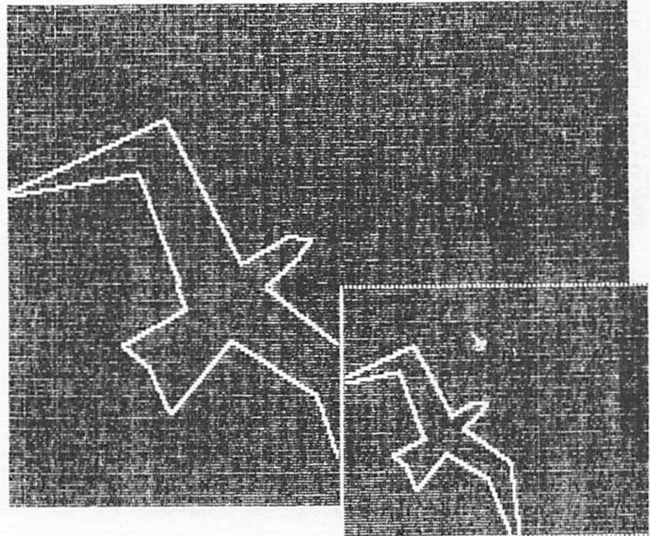
I find it unnecessary (and often almost impossible) to step thru all the levels in planning a recursive procedure; if one reads the toplevel procedure as a "user" of the action-of-the-whole at each of its **MYSELF**s, or reads **MYSELF** as the typical interface between two levels, that should be enough for understanding. But how many levels deep might it actually go in the search for the viewport boundary? To determine the risk of return stack overflow I simulated a high-resolution (1024 wide) graphics display and clipped random lines from a large user space 30000 pixel units wide. Interestingly, and reassuringly, the depth of recursive calls NEVER exceeded 17 — staying well within both parameter stack and return stack limits. Of course a high-level recursive search is slow; the real efficiency of this algorithm would be realized in assembler CODE, calculating the midpoints with two additions and two right shifts.

To illustrate uses of clipping, I have included printouts of a decorative title for this article and a frame from an animation sequence in which a bird and its reduced likeness fly into and out of large and small viewports simultaneously on the screen. □

BIBLIOGRAPHY

1. Foley, J.D. and Van Dam, A., *Fundamentals of Interactive Computer Graphics*. Addison-Wesley Publishing Company, 1982.
2. Newman, W.P. and Sproull, R.F., *Principles of Interactive Computer Graphics*, McGraw-Hill, 1979.

Bob Gotsch is a graphics programmer for Time Arts, Inc. and a teacher of graphic arts at the California College of Arts and crafts. He is interested in exploring the use of computers as aids to artists. He uses FORTHWARE FORTH.



FORTH Based File Handling System

(continued from page 9)

The ability of having several channels active allows easy file to file transfer of information, or simultaneous editing of several files. However, since it would then be possible to have several blocks with the same block number yet on different channels, the routines like **BLOCK** need some very minor alterations to prevent confusion. If there is enough interest from readers, I will discuss these changes in a future article. For now, I recommend that **CHAN** be ignored, and that all file I/O be performed on the default channel zero, and that files be opened, used and closed sequentially.

Conclusion

A FORTH that has the file handling capability has many advantages. The one illustrated is simple, requiring only file string, FDB stuffer and three verbs. Error recovery is as simple. Yet it clarifies FORTH program usage by making source code more modular and circumventing much code since there is no need for documentors or auto-indexes. The effort to add such to FORTH is trivial, due to the modest amount of additional code. The gain is easy file generation, be it FORTH source code, formatted text, target compiled FORTH object code or FORTH generated executable code. □

FORTH Standards Corner

Towards a New Standard

Robert L. Smith

The Standardization Effort

The FORTH Standards Team met in May at the National 4-H Center, Washington, D.C. The team decided to work towards a new FORTH Standard, tentatively called FORTH-83. The mechanism of producing a FORTH Standard seems to be evolving. Previously the team members met, discussed, and then voted on a variety of topics during a three day session. Ambiguities and smooth wording were worked out by a smaller group of referees. The resulting document was then offered for acceptance as a whole by at least two-thirds of the voting members. One of the problems with this method is that the time for deliberation is far too short for the proposals and implications to be thoroughly understood.

The next Standard will evolve through various working drafts. It is intended that there be opportunities for public examination and input. By having more than one meeting prior to acceptance of the next Standard, changes and corrections can be made which should reduce the inconsistencies in the final document.

Perhaps the future standardization efforts should be split into separate functions somewhat like that of the COBOL or MUMPS standardization committees. At the lowest level a language development committee meets regularly to make changes to the language. Their output is published as a journal, for consideration and testing by implementers and users. A separate Standards committee generates an actual Proposed Standard document on the time scale of five years. They freeze the output of the language development committee (who continue to work independently). After a suitable voting process, this document becomes the new Standard. Thus the community of users has an adequate chance to make their views known. Additional steps in the process involve approval by an ANSI committee, then perhaps other governmental or quasi-governmental committees.

Vocabularies

The area of greatest concern for the next Standard is that of vocabularies. FORTH-79 has a very weak vocabulary structure. It was chosen as the minimum subset of most FORTH implementations. The only weaker structure is the complete lack of vocabularies in older versions of FORTH, such as DECUS or OVRO (Cal Tech) FORTH. In FORTH-79, the search order at a given time is through two vocabularies: the one specified by CONTEXT and then the FORTH vocabulary. Some other FORTH systems (like fig-FORTH) have vocabularies linked together in a tree structure determined when the vocabularies are created. The search order is determined by the vocabulary last activated and its predecessors in the tree structure down to the trunk of the tree (which is usually FORTH). In poly FORTH systems the search order when a given vocabulary name is invoked is determined by a four nybble (in one word) parameter given when the vocabulary name is created. Typically this limits the total number of separate vocabularies to 7 or possibly 15.

A dynamic method for determining search order uses the "vocabulary stack." This is a concept taken from STOIC. Each wordset is "sealed," i.e., not linked to any other. A wordset is pushed onto the vocabulary stack from, say, the value in CONTEXT by using a word such as VPUSH (my favorite name for this function is ALSO). Another word is used to drop the top member of the vocabulary stack, or perhaps to clear it out entirely. Bill Ragsdale uses the word ONLY for the latter purpose. By first searching CONTEXT and then the vocabulary stack we can maintain a reasonable amount of upward compatibility. This is an idea advanced by George Shaw at the last FORML Conference.

There are many other possibilities. Don Colburn has suggested a defining word like SEARCH-ORDER which would name a word which specifies the search order. John James has suggested that the invocation of a vocabulary or wordset name would push itself onto the vocabulary stack if it was not currently on the vocabulary stack. Otherwise the stack would be truncated back to its first appearance on the stack.

There are other designs for vocabulary mechanisms. Almost any of them would be an improvement over FORTH-79. In my opinion it is important that the next Standard have a significant improvement in vocabulary structures. If you have any strong opinions on this matter, please communicate them in writing to the FORTH Standards Team.

FORTH Standards Team Upcoming Working Meeting

A working session in the development of the FORTH language 1983 standard (FORTH-83) has been scheduled this October 3rd through 5th in Carmel Valley, California. Space will be limited with priority for existing standards team members. Accommodations will cost US$150 based on double occupancy including meals. Room reservations require a deposit of US$50 and should be received by July 31.

This working session will attempt to resolve the FORTH-83 Standard working draft in anticipation of an accepted standard near the beginning of 1983. This working draft will be available for US$15 beginning August 1. Comments on this working draft are encouraged. Standards team sponsors additionally receive all mailings to team members prior to the October meeting, including copies of submitted proposals and comments. Standards team sponsorship is available for US$50.

Please send orders, deposits or inquiries directly to the FORTH Standards Team, P.O. Box 4545, Mountain View, CA 94040, USA; or telephone Mr. Roy Martens at (415) 962-8653.

FORTH-83 DO-LOOP

Robert L. Smith

A new form of the **DO-LOOP** has been accepted for the next FORTH standard, tentatively called FORTH-83. The new **DO** will generally work as you would expect for indices which represent either addresses or signed or unsigned arithmetic values. The index **I** covers a complete 65K range, the same as in FORTH-79 but twice as much as in fig-FORTH or poly-FORTH. An additional advantage occurs with **+LOOP**: the sign of the increment can change within the loop without necessarily causing an exit condition. The speed of the new form is faster than most previous loops unless **I** occurs frequently. A feature of the new loop is that when **LEAVE** is executed, control is passed to the end of the loop without intervening calculations.

The price to be paid for the general form of the new loop is that certain "side-effects" of the old form are missing. Consider the simple definition:

: TEST 0 DO I . LOOP ;

Under the old form, **-5 TEST** would execute exactly once. In the new form, the loop continues until the index **I** crosses the boundary between limit and limit-1. In the above case, **-5 TEST** would print out:

0 1 2 ... 32767 -32768 -32767 ... -8 -7 -6

To print only one value would require **1 TEST**. For another example, consider the following function:

: CLEAR DO 0 I C! LOOP ;

Suppose that our base is hexadecimal and we wish to clear memory between 2000 and EFFF. With the new form of **LOOP**, we could simply type:

F000 2000 CLEAR

and the indicated area would be cleared. The routine would only clear one byte with the FORTH-79 or the fig-FORTH version of **LOOP**.

The new loop considers that the index **I** lies on a "number circle" based on the usual 2's complement arithmetic for 16 bit numbers. Thus there is a smooth transition between −1 and 0 and between 7FFF and 8000 hex (32767 and −32768 decimal).

There are a variety of ways to implement the new loop, some of which remain to be discovered. There are two parts to the problem. One is to find a method of calculating the exit conditions, and the other is to allow **LEAVE** to work properly. The fastest method of determining the exit conditions requires that the actual value of **I** be calculated by an addition or subtraction. The items stored on the return stack (or elsewhere) are related to the limit and the index, but are not necessarily the same. For machines with a testable overflow bit the suggested technique is to modify the limit and initial index so that the transition will lie between 7FFF and 8000 hex. The overflow bit is set whenever an addition causes the result to cross the 7FFF to 8000 boundary. Initially put

limit' <-- limit + 8000
I' <-- init - limit'

To calculate **I**, note that

I = I' + limit'

At **+LOOP**,

I' <-- I' + increment

Then check for overflow. If the overflow bit is set, continue to loop, else exit from the loop.

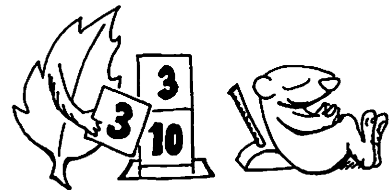
For machines without an overflow bit, such as the 8080, let

limit' <-- limit
I' <-- init - limit'

For **LOOP**, merely increment **I'** by 1 and branch back if the result is non-zero. For **+LOOP**, one has to examine the combination of the carry bit and the sign bit of the increment. Klaus Schleisiek suggests the following: use the RAR instruction to shift the carry bit into the accumulator, then use XRA with the increment value. Only the sign bit of the result is of interest. If the result is positive, continue to loop. If the result is negative, terminate the loop.

There have been various suggestions for implementing **LEAVE**. Bob Berkey's original suggestion involves having the run-time operator for **DO** place the exit address for the loop on the return stack to be used by **LEAVE**. Klaus Schleisiek improved that by having **LEAVE** be an immediate word. By using the return stack at compile time to store the addresses of "fixup" locations, it is possible to avoid run-time penalties when **LEAVE** does not occur in the loop. Bill Ragsdale has suggested a simple but clever way of avoiding use of the return stack (except as a very temporary storage place), since Klaus's method may not be compatible with certain systems. Bill's technique links the forward references in a simple chain and then resolves the chain when the **LOOP** or **+LOOP** is encountered at compile time. As a result of his work, Bill has also suggested an alternative form of **LEAVE**, called **?LEAVE**, which appears to be more useful than **LEAVE** itself. **?LEAVE** takes the top element from the parameter stack and terminates the loop if the element is non-zero (i.e., true). Further details will probably be presented at the next FORML meeting.

One interesting possibility for augmenting the new **DO** is to add a function called, say, **?DO**. When the arguments to **?DO** are equal, as in the case **0 0 ?DO**, then the loop is not executed at all. If that appears sufficiently useful, then that function could be incorporated in **DO** itself, so that an additional word would not be needed. It would require a slight amount of additional time at the beginning of each loop, and would eliminate one (admittedly rarely used) case from **DO**. ◻



Reprinted from Starting FORTH, by Leo Brodie, permission of Prentice-Hall, Inc.

Fig Chapters

U.S.

• ARIZONA

Phoenix Chapter
Dennis L. Wilson, Samaritan Health Services, 2121 E. Magnolia, Phoenix, AZ, 602/257-6875

• CALIFORNIA

Los Angeles Chapter
Monthly, 4th Sat., 11 a.m., Allstate Savings, 8800 So. Sepulveda Blvd., L.A. Philip 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., 12 noon, Fullerton Savings, 18020 Brookhurst, Fountain Valley. 714/523-4202

San Diego Chapter
Weekly, Thurs., 12 noon. Call Guy Kelly, 714/268-3100 x4784

• MASSACHUSETTS

Boston Chapter
Monthly, 1st Wed., 7 p.m. Mitre Corp. Cafeteria, Bedford, MA. Bob Demrow, 617/688-5661 after 5 p.m.

• MICHIGAN

Detroit Chapter
Call Dean Vieau, 313/493-5105

• MINNESOTA

MNFIG Chapter
Monthly, 1st Mon. Call Mark Abbot (days) 612/854-8776 or Fred Olson, 612/588-9532, or write to: MNFIG, 1156 Lincoln Ave., St. Paul, MN 55105

• MISSOURI

St. Louis Chapter
Call David Doudna, 314/867-4482

• NEVADA

Las Vegas Chapter
Suite 900, 101 Convention Center Dr. Las Vegas, NV 89109, 702/737-5670

• NEW JERSEY

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

• NEW YORK

New York Chapter
Call Tom Jung, 212/746-4062

• OKLAHOMA

Tulsa Chapter
Monthly, 3rd Tues., 7:30 p.m., The Computer Store, 4343 So. Peoria, Tulsa, OK. Call Bob Giles, 918/599-9304 or Art Gorski, 918/743-0113

• OHIO

Dayton Chapter
Monthly, 2nd Tues., Datalink Computer Center, 4920 Airway Road, Dayton, OH 45431. Call Gary Ganger, (513) 849-1483.

• OREGON

Portland Chapter
Call Timothy Huang, 9529 Northeast Gertz Circle, Portland, OR 97211, 503/289-9135

• PENNSYLVANIA

Philadelphia Chapter
Call Barry Greebel, Continental Data Systems, 1 Bala Plaza, Suite 212, Bala Cynwid, PA 19004

• TEXAS

Austin Chapter
Call John Hastings, 512/327-5864

Dallas/Ft. Worth Chapter
Monthly, 4th Thurs. 7 p.m., Software Automation, 1005 Business Parkway, Richardson, TX. Call Marvin Elder, 214/231-9142 or Bill Drissel, 214/264-9680

• UTAH

Salt Lake City Chapter
Call Bill Haygood, 801/942-8000

• VERMONT

ACE Fig Chapter
Monthly, 4th Thur., 7:30 p.m., The Isley Library, 3rd Floor Meeting Rm., Main St., Middlebury, VT 05753. Contact Hal Clark, RD #1 Box 810, Starksboro, VT 05487, 802/877-2911 days; 802/453-4442 eves.

• VIRGINIA

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

• WASHINGTON

Seattle Chapter
Call Chuck Pliske or Dwight Vandenburg, 206/542-7611

FOREIGN

• AUSTRALIA

Australia Chapter
Contact Lance Collins, 65 Martin Rd., Glen Iris, Victoria 3146, or phone (03) 292600

• CANADA

Southern Ontario Chapter
Contact Dr. N. Solntseff, Unit for Computer Science, McMaster University, Hamilton, Ontario L8S 4K1, 416/525-9140 x2065

Quebec Chapter
Call Gilles Paillard, 418/871-1960 or 643-2561

• ENGLAND

English Chapter
Write to FORTH Interest Group, 38 Worsley Rd., Frimley, Camberley, Surrey, GU16 5AU, England

• JAPAN

Japanese Chapter
Contact Masa Tasaki, Baba-Bldg. 8F, 3-23-8 Nishi-Shimbashi, Minato-ku, Tokyo, 105 Japan

• NETHERLANDS

HCC-FORTH Interest Group Chapter
Contact F.J. Meijer, Digicos, Aart V.D. Neerweg 31, Ouderkerk A.D. Amstel, The Netherlands

• WEST GERMANY

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

SPECIAL GROUPS

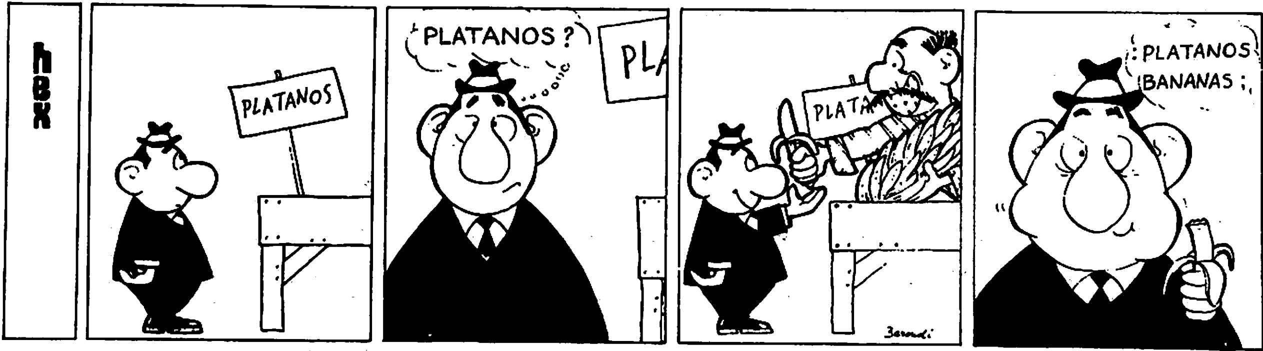
Apple Corps FORTH Users Chapter
Twice monthly, 1st & 3rd Tues., 7:30 p.m., 1515 Sloat Blvd., #2, San Francisco, CA. Call Robert Dudley Ackerman, 415/626-6295

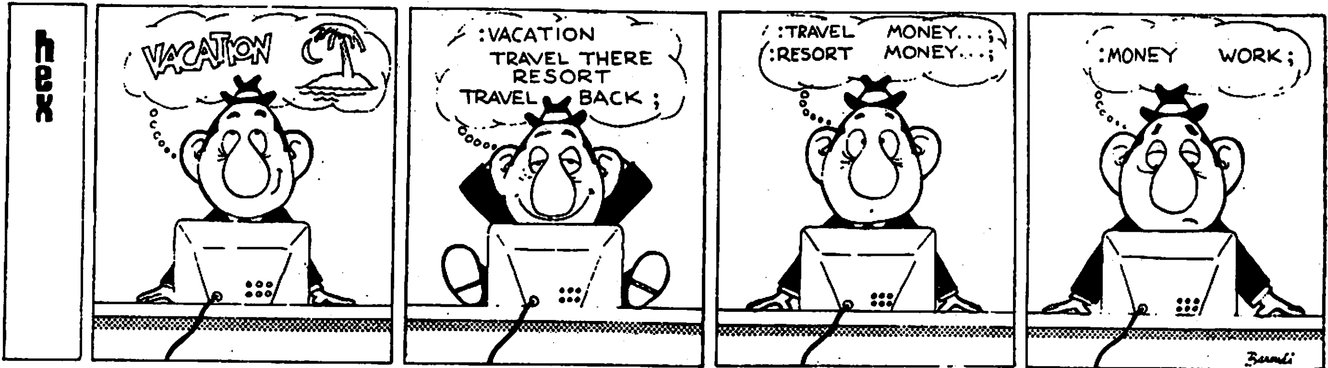
Detroit Atari FORTH
Monthly, 1st Wed.
Call Tom Chrapkiewicz
313/524-2100 or 313/772-8291

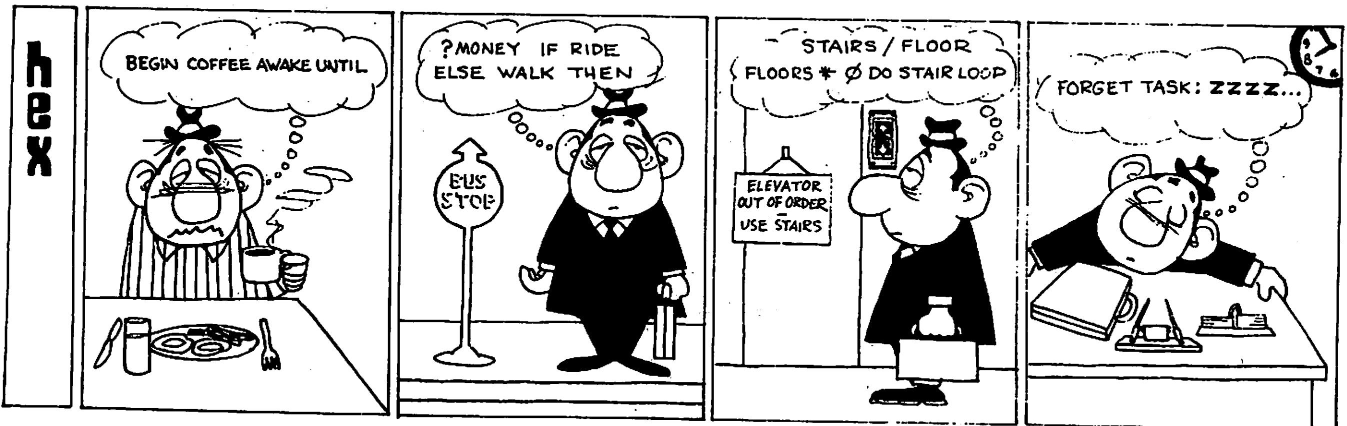
Nova Group Chapter
Contact Mr. Francis Saint, 2218 Lulu, Witchita, KS 67211, 316/261-6280 (days)

MMSFORTH Users Chapter
Monthly, 3rd Wed., 7 p.m., Cochituate, MA. Dick Miller, 617/653-6136

CARTOONS







LEARN FORTH
JOIN FIG!

The best way to learn FORTH and keep up with implementation and application information is to join the FORTH Interest Group. You will receive each issue (six) of FORTH Dimensions as it is published and you will be able to join a local FIG Chapter.

Membership costs $15.00 US, or $27.00 Foreign and runs concurrent with the magazine year. Volume V covers from May, 1983 through April, 1984. Back Volumes I, II, III, and IV are available for $15.00, US or $18.00 Foreign.

☐ **Yes,** I want to join FIG and receive all of Volume V.

Name ______________________________

Organization ______________________________

Address ______________________________

City ______________________________

State ______________________ ZIP __________

VISA ______________________________

MASTERCARD ______________________________

Expiration Date for charge card: ______________________________

Make check or money order in US Funds on US Bank, payable to **FIG.** All prices include postage. California residents add sales tax except on current membership. No purchase orders accepted without checks.

ORDER PHONE NUMBER: 415/962-8653

FORTH INTEREST GROUP PO Box 1105 San Carlos, Calif. 94070

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 ☐ 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)	$16	$20
☐ "Starting FORTH" by Brodie. (Hard Cover)	$20	$25
☐ PROCEEDINGS 1980 FORML (FORTH Modification Lab) Conference	$25	$35
☐ PROCEEDINGS 1981 FORML Conference, Both Volumes	$40	$55
☐ Volume I, Language Structure	$25	$35
☐ Volume II, Systems and Applications	$25	$35
☐ PROCEEDINGS 1982 FORML Conference	$25	$35
☐ PROCEEDINGS 1981 FORTH Univ. of Rochester Conference	$25	$35
☐ PROCEEDINGS 1982 FORTH Univ. of Rochester Conference	$25	$35
☐ FORTH-79 Standard, a publication of the FORTH Standards Team	$15	$18
☐ Kitt Peak Primer, by Stevens. An in-depth self-study primer.	$25	$35
☐ BYTE Magazine Reprints of FORTH articles, 8/80 to 4/81	$ 5	$10
☐ FIG T-shirts: ☐ Small ☐ Medium ☐ Large ☐ X-Large	$10	$12
☐ Poster, August 1980 BYTE cover, 16" x 22"	$ 3	$ 5
☐ FORTH Programmer's Reference Card. If ordered separately, send a stamped, addressed envelope.	FREE	

TOTAL $__________

NAME______________________________ MAIL STOP/APT______________

ORGANIZATION______________________ PHONE (   )______________

ADDRESS____

CITY__________________ STATE__________ ZIP________ COUNTRY__________

VISA #____________________ MASTERCARD #______________________

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. 1/83

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