Volume 6, Number 6

# FORTH Dimensions

March/April 1985
$2.50

## The Far Right Stuff

Local Definitions

In-Word Parameter Words

TI 99/4A Screen Dump

Mathquiz

Enhanced DO LOOP

# FORTH IS NOW VERY.FAST!

.Sieve 1.3s/pass
.Compile 300 screens/minute
.Drop 1.82 us
.Concurrent I/O @ 250K baud

## DEVELOP YOUR APPLICATIONS IN A TOTAL FORTH ENVIRONMENT.

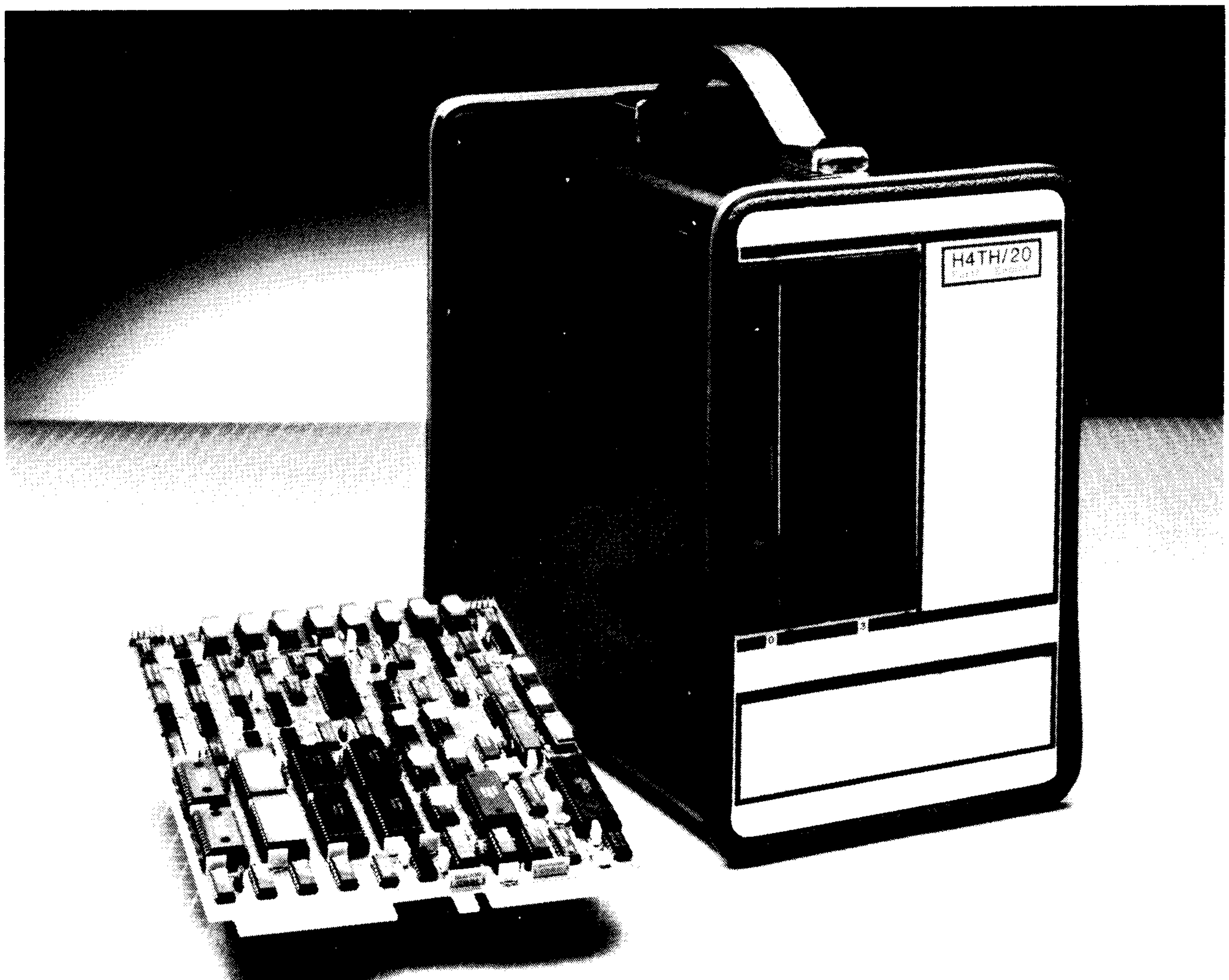
**MICROPROGRAMMED BIT SLICE FORTH ENGINE**
.Microcoded forth kernel
.Microcoded forth primitives
.Multi-level task switching architecture for real time applications
.Optional writable control store

**H.FORTH OPERATING SYSTEM**
.Hierarchical file system
.Monitor level for program debug
.Multi-user multi-tasking
.Target compiler
.I/O management
.Forth 83 Compatible

**H4TH/01 OEM SINGLE BOARD**
.Floppy disk controller
.2 channel SIO to 38.2K baud
.Calendar clock—4HR backup
.44K Byte ram 200NS
.32K Byte EPROM operating system
.1K X 32 microprogram memory 70ns

**H4TH/10 DESKTOP**
.Dual 0.8m Byte floppys
.H4TH/01 processor
.Three user slots
.Two expansion slots
.Power & cooling

**H4TH/20 DESKTOP**
.10 m Byte Winchester
.0.8 m Byte floppy
.H4TH/01 processor
.300K byte RAM expandable 2m byte
.Three user slots
.One expansion slot
.Power & cooling




A forth-engine consisting of a state-of-the-art integrated hardware/software system giving unsurpassed performance for professionals and their applications from a company that is totally dedicated to the forth concept and its implementation.

**HARTRONIX, Inc.** 1201 North Stadem Drive Tempe, Arizona 85281 602.966.7215

**FORTH Dimensions**
Published by the
Forth Interest Group
Volume VI, Number 6
March/April 1985
Editor
Marlin Ouverson
Production
Cynthia Lawson

*Forth Dimensions* solicits editorial material, comments and letters. No responsibility is assumed for accuracy of material submitted. Unless noted otherwise, material published by the Forth Interest Group is in the public domain. Such material may be reproduced with credit given to the author and to the Forth Interest Group.

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 to submit material for publication, the address is: Forth Interest Group, P.O. Box 8231, San Jose, California 95155.

## Symbol Table

Simple; introductory tutorials and simple applications of Forth.

Intermediate; articles and code for more complex applications, and tutorials on generally difficult topics.

Advanced; requiring study and a thorough understanding of Forth.

Code and examples conform to Forth-83 standard.

Code and examples conform to Forth-79 standard.

Code and examples conform to fig-FORTH.

Deals with new proposals and modifications to standard Forth systems.

# FORTH Dimensions

## FEATURES



## DEPARTMENTS

# Letters to the Editor

### Hang Ten for Lunch

Dear Mr. Ouverson:

I thought you and other members of FIG might like to see some examples of a language I read about recently:

SPEAKER SURFBOARD FETCH
WINDOW TAIL-TOUCH
BALL QUESTION
SURFBOARD RIGHT FRISBEE FETCH

No, it isn't Forth, although one of its principles is, "... object words precede action words, and modifiers come before objects." According to *Science News* (Dec. 1, 1984), this is one of two languages researchers at the Kewalo Basin Marine Mammal Laboratory in Honolulu use to communicate with dolphins. With this language, the "words" are hand gestures. The other language, which is not so Forth-like, uses computer-generated whistles.

Yours truly,
Bob Lewis
Portland, Oregon

### Passport for the STATE-less

Dear Marlin,

fig-FORTH and Forth-79/83 have a user variable called **STATE** which contains the compilation state. A non-zero value means that compilation is occurring. Some words, such as **LITERAL** in fig-FORTH, test this user variable and execute differently depending on its value. This is known as state-smartness. However, there appears to be a movement afoot to eliminate state-smartness. This is mostly a good thing: it keeps life simple if a word does the same thing when processed by the compiler as it does when processed by the interpreter. However, translating routines dependent on state-smartness into state-less Forth dialects can be complicated.

An example of state-smartness dependency is Alden Long's number conversion routine ("Letters," *Forth Dimensions* VI/2). Numbers lead a double life. When processed by the compiler they are stored into the dictionary as a literal. When processed by the interpreter they are left on top of the stack. Thus, any word which does its own number conversion requires state-smartness.

One way of emulating the **STATE** user variable is to test the smudge bit of the most recent word in the dictionary. Most Forth dialects use a smudge bit on a word's header to hide an incompletely compiled word from a dictionary search. If the smudge bit of the last dictionary entry is set, it normally means we are compiling the current word. If the smudge bit of the last word in the dictionary is clear, the word is complete and, therefore, we must be in the interpreter. Unfortunately, there is a case when this doesn't work. If you use **[ ]** to break out to the interpreter while compiling, the smudge bit is set at that point.

Although fig-FORTH has a **STATE** user variable and a state-smart **LITERAL**, here is an example of how it can be emulated with a word **COMPILING?** which returns the value of the latest smudge bit. fig-FORTH has a word **LATEST** which returns the name field address of the latest dictionary entry in the **CURRENT** vocabulary. From it, we can get the smudge bit and test it.

```
: COMPILING? ( — t | f )
  LATEST C@ 32 AND ;
```

PolyFORTH, which I use, has a similar word **LAST** which is a variable containing a pointer to a pointer to the last word in the dictionary, hence:

```
: COMPILING? ( — t | f )
  LAST @ @ 2+ C@ 32 AND ;
```

And finally, an example of how you would use this word to emulate a state-smart **LITERAL**

```
: SMART-LITERAL ( n — [n] )
  COMPILING?
  IF [COMPILE] LITERAL
  THEN ; IMMEDIATE
```

Yours sincerely,
Paul Bigelow
Waterloo, Ontario

### Already Abstruse

Dear Mr. Ouverson,

I just received my renewal notice today, and I noticed the change in editorial policy, with its heavier emphasis on applications and practical tips. I want to commend you for the change. For some time now, I have believed that the reason Forth is not more widely used is because it has not had a journal willing to address itself to novices. Forth is abstruse enough by itself; it needs a presentation to beginners that is clear and practical. I'm sure that if the very real virtues of the language can be clearly demonstrated to beginning programmers, it will quickly win a much wider following.

Sincerely,
Scott Locicero
Las Vegas, Nevada

*All right, Forth authors, time to get busy! We aim to serve beginners as we continue to delve into the advanced mysteries. Don't be shy of "heavy" subjects or negligent of others which seem trivial; but do some comparative research of similar ideas, define all your words, document your screens well, avoid system dependencies and present plenty of text in proportion to the amount of code. FIG member David A. Ryder wrote from Michigan last year, "I do not wish to seem bitter, but to me the problem is not with portability. It is a matter of communication. I am hoping the next volume will help a rank beginner like myself." Our editorial files have plenty of similar letters, but we need a fresh reserve of material to meet this special need. Who will champion the novices' cause?*

*—Marlin Ouverson*

# Editorial

# Xie, Xie

Time does fly when you're having fun, and it's already time to remind you to renew your FIG membership. Many have already done so, ensuring uninterrupted delivery of *Forth Dimensions* and other important material. Plans are under way to expand member services in some dramatic ways, and we hope each and every one of you will share in the benefits. To one and all, thank you for supporting FIG's activities during past years.

We would also like to recognize once more all of the gracious overseas organizers and sponsors of the 1984 FORML China Tour. The fruit of their labors was a series of successful conferences in the Republic of China (Taiwan) and the People's Republic of China (mainland). Our coverage of last year's tour concludes in this issue. Every conference was well organized and was well attended by serious and enthusiastic audiences, the kind we were proud to be part of. Our thanks and appreciation go to all who were involved with these functions!

If you stayed home last year and missed the Forth techniques and important accomplishments discussed at the FORML conferences, all is not lost. FIG members in West Germany are putting together an exciting FORML event for which they are soliciting papers, speakers and attendees. We will keep you up to date as details are given to us — check for more in "Products & Announcements."

The event of note in this issue can be found in the "Techniques Tutorial." Henry Laxen has tossed down a gauntlet that few may dare to pick up: the Mystery of the Grand Case is waiting to be solved, and you have only one issue's time to do it. Those of you who have already renewed your FIG membership can go directly to Henry's column; all others have another priority — if you want to read about the solution in Volume Seven, Number One.

— *Marlin Ouverson*

---

**Missing Enigma**

Dear Fellow FIG-ers:

I like the new "look" of *Forth Dimensions*, but miss the enigmatic definitions of Al McCahon. Can't you bribe or threaten him from time to time for even a little tidbit? If possible, would you consider printing regularly an index of Forth-related articles from other publications?

Keep up the good work!

Sincerely,
Danny K. Liles
Jackson, Mississippi

*I'll have a talk with Al soon... As for the index, the best source is the* Bibliography of Forth References, *recently updated and expanded. When you see how many references it contains, you'll understand why it doesn't appear in our magazine pages. It is published by the Institute for Applied Forth Research, Inc., and can be found as well on the FIG Order Form in* Forth Dimensions.

— *Marlin Ouverson*

**Color Forth**

Greetings,

Knowing that FIG is always interested in fig-FORTH implementations on new machines, I would like to take this opportunity to inform the membership of one such.

Last summer I was able to get 6800 fig-FORTH running on the Radio Shack MC-10 Micro Color Computer, which has a 6803 MPU. I had to rewrite 6800 Forth to run in high memory and, while I was at it, I fixed a couple of minor bugs and added some extensions necessary to make fig-FORTH useful on this tape-based system. Incorporated are: a **CSAVEM** command to save a block of memory to tape, and commands to plot high-resolution graphics (the MC-10 lacked these commands); **PRINTER** and **SCREEN** commands to direct output to either the console or a printer; a **SOUND** command for creating music; and Forth screens are simulated in RAM. This implementation, which I call MC-FORTH, requires the MC-10 to have the Radio Shack 16K RAM pack, for a total of 20K of memory.

I have used MC-FORTH for a number of months now, and have had a wonderful time using it to learn about Forth. The offer I would like to make to all members of FIG is this: if you would like me to send a tape to you containing this Forth for the Radio Shack MC-10 Micro Color Computer, the fig-FORTH line editor and some assorted Forth utilities (recursive decompiler, dump, etc.), send $15 to cover my time and materials, and your name and address. Then you'll finally be able to do something interesting with your MC-10!

Again, thank you for your efforts on behalf of all of us FIG-ers, and I hope my offer helps a few.

Thank you,
Gobind Singh Khalsa
Liverpool, New York

# Learning Forth

*William F. Ragsdale*
*Hayward, California*

"Ask the Doctor" is Forth Dimensions' *health maintenance organization to aid your understanding and use of Forth. Questions of a problem-solving nature, on locating references, or just regarding contemporary techniques are most appropriate. When needed, your good doctor will call on specialists. Published letters will receive a preprint of the column as a direct reply.*

In his last column, the good doctor addressed the common problem of organizing one's approach to learning Forth. The components of attitude, computer, version and documentation were addressed. This month, your faithful practitioner expands on self-study, based on a new book.

We respond to the letters received from Bill Hepner of Anaheim, CA; John Megar of Snellville, GA; Dennis Miller of Weymouth, MA; Louis Lebovitz of Queens Villiage, NY; and Isadore Nicholson of Greenvale, NY. These readers ask for suggestions on learning Forth with systems accompanied by modest documentation. Since the earlier column appeared, a book specifically intended for self-paced learning has been published.

Because learning Forth is such a common topic in your Forth doctor's mailbox, he will devote this full column to the new book *Learning Forth* by Margaret A. Armstrong with technical assistance from Dr. Mitchell E. Timin. This volume was published in 1985 by John Wiley & Sons, Inc., New York; 226 pages, softbound, $16.95.

The material is organized into eleven chapters, five appendices and an index. The chapters, although titled differently, are organized as follows:

1. Why you should program, motivation
2. Forth history
3. Start-up
4. Stack use, number representation
5. Defining new words
6. Style and structure
7. Editor
8. Conditiona and Booleans
9. Data structures
10. Interactive examples

The foreword states the author's purpose to develop student competency with "...a basic working knowledge of Forth..." and "...to be ready to move on to intermediate-level programming in Forth." In summary, Armstrong accomplishes her goal with satifying directness.

The Socratic style of mixing questions, answers and new material offers the reader a structured approach without being repetitive. Each concept is presented in a brief paragraph, with several one-line questions. Thus, 90% of the book presents "frames of knowledge" with requests for reader involvement. In fact, since text, questions and answers alternate, the book would be quite hard to read in narrative fashion. Each chapter concludes with several summary questions.

A suggestion is in order. You will lose much of the "dialog" format of the book if you read ahead to the answer before truly thinking through each question. Try using a 3x5 card to cover the answers until you have had a chance to formulate your own. It's fine to look back at the text, just resist any temptation to read the answers! Even better, write down your answers. This is involving, and will later reveal your true progress and understanding.

## Good Points

The first chapter succeeds at engaging the reader with discussion of the benefits of programming, as well as presenting products from a number of vendors. A brief history of Charles Moore's work sets the stage for the later, technical material.

A few inaccuracies about the early origins of Forth and FIG are to be found, indicating the author worked from published sources rather than from first-hand research. These do not detract from the appropriateness of historical perspective.

The topic of text entry to disk is treated in a functional way rather than in a technical way. The reader learns how to list and load from disk, but must get the specifics of editing from his system documentation. This is out of consideration for the reader, since subtle differences are found in editors purported to be identical. Keen insight or good fortune also caused the author to omit other elements of Forth usage that have variations in use or subtlety in application. Words such as **BLOCK**, **VOCABULARY** and **DOES>** are thankfully absent, as they can intrude on the initial development of competency.

Many of the examples build to a major project. Chapter Eleven calls on these components when presenting the example of a simple computerized payroll. All the components of a "real" application are present, although offered in a limited way. We see data structures, prompted input, calculation and a report. This is the first time in print for a coordinated use of such elements to teach Forth. An appendix also gives twenty-four blocks of short application examples.

Systems other than Forth-79 or fig-FORTH will have trouble with the examples using **BL**, **WORD**, **QUERY** and **SP@**. However, such dependencies are far fewer than in other texts.

At first glance, this book is similar to many other books on Forth. But, looking deeper, we see an instance of the 80–20 Pareto principle. Other books spend only twenty percent of their space on the twenty percent of Forth that is the core of usage. The remaining eighty percent is spent on completeness, details and side effects.

Armstrong turns around the ratio. She spends eighty percent of the book on the twenty percent of Forth you use eighty percent of the time. Some of the stack manipulation examples are lengthy; the time you devote to this material will be well spent if you truly master it.

### Problems

All topics are presented at the same level of emphasis. This tends to mask some items of great importance. For example, on page twenty-five, the text interpreter is explained in five lines. The explanation given is not rigorous, and is buried without a caption as the fifteenth point in the chapter.

This points out a subtle risk to the reader. Since information is presented in small bites, one may jump over and skip ahead. The risk is that impatient students will succumb to the temptation to skip ahead. Still, encapsulation of information is essential to the learning process, even if the technique is not appreciated by the student. It has been stated that "you can only learn what you almost know." The building-block approach is used here; but without an instructor present, the burden of pacing is relegated to the student.

Some fundamentals need to be stated clearly the same way every time. There is some confusion between "a word" and "a word definition." Also, no references are cited in this work. And Armstrong's book would have benefitted from a clearer correlation with publications such as the Forth-79 and Forth-83 Standards.

### Philosophy and Style

Important methods of top-down design, natural-language problem statement and successive refinement are brought into play quite early (beginning on page three, to be exact).

But no help is offered on screen editing style. There is not even a sample screen in the body of the book, although there are many in Appendix A. This problem is brought home on page 121, where a single definition is thirty-three lines long. On page 124, insult is added to this injury when the same example is expanded to forty-eight lines! This would have been a great place to discuss factoring and modularization.

Only one place in the book offers mystification to readers. The fig-FORTH and Forth-79 **EXPECT** will leave one or two nulls at the end of input text. This is mentioned in the book's glossary. Several examples use **VARIABLE** and **ALLOT** to leave space for text and numbers. It is a side effect of combined use that two extra bytes are allocated. This space nicely allows for the nulls, but no mention is made of such a need. If a student chose to conserve the space, a system crash would be likely. A more general technique would be to allot a scratch space, clear it to blanks, then **EXPECT** and **CMOVE** the desired characters to the storage areas.

### Terminology

No mention is made of the Forth dialect used in the examples. The glossary is fig-FORTH, and it appears that all the examples will work in fig-FORTH and Forth-79. Five words are used which are not in Forth-83, but they may be provided in extensions by specific vendors (**VLIST**, **BL**, **QUERY**, **SP@** and **SP!** ). It is a shame that simple overlays were not given, so that users of most distributed systems could make immediate use of this book. The terminology is quite up to date. Although the dialect appears to be fig-FORTH, the effects of Forth-79 and Forth-83 show in the text. Terms such as "text interpreter" are used, rather than the dated "outer interpreter." Thus, readers will be learning contemporary expression.

The author does have the abominable habit of capitalizing the word Forth. This makes reading difficult, and continues the confusion between Forth as a proper noun (it is not an acronym), trademarks built on the name, and the word **FORTH** as the name of a vocabulary when in the dictionary.

### What This Book is Not

Do not expect to use this book as a reference manual. English topic headings are at a premium and the index is minimal. Each major paragraph or subject is numbered, but no use of the numbers is apparent. A keyword would have been more user-oriented. While chapters are logically laid out, you will generally have to search linearly within a chapter to return to a specific subject.

The material on each topic is clear, usable and understandable. No attempt is made to be comprehensive or to discuss side effects. As a result, the material should be transportable to a variety of Forth systems. For example, **LEAVE** is not discussed during consideration of **DO** and **LOOP**. This is probably better left for other works, as compatibility problems and side effects are likely. In a similar fashion, discussion of the use of the disk for other than storage of programs is left alone.

### Conclusion

Your good doctor is aware of no other Forth book which is offered solely as a learning experience. Other authors have written books from which they wish one to learn, but here is the first chance for you to participate completely in such an activity. Congratulations are in order for Margaret Armstrong for producing *Learning Forth.*

# In-Word Parameter Words

*Sam Suan Chen*
*Lung Tan, Taiwan*
*Republic of China*

In a short note in *Forth Dimensions* (V/3), Dr. Timothy Huang used the in-word parameter technique to implement three new words: **DUPx**, **DROPx** and **SWAPxy**. These three words are quite fancy. However, they can be used only in interpretive mode, not in colon definitions. For example, a word such as **Z** (screen 53) will not work; even its compilation cannot be completed normally.

Here we present an improvement to Huang's scheme. The words used to achieve our goal are given on screen 54. They are all 79-Standard except **SP@** and **WIDTH**. Although non-standard to Forth-79, these two words had been

```
 OK
55 LOAD 53 LOAD
2 4 1 3
2 4 1 3  OK

 OK
' Z DISM

ADDR   CODE
----   ---- --
72EF-  71D5 CR
72F1-  0FBB 1
72F3-  0FC3 2
72F5-  0FCB 3
72F7-  080E (LIT)
72F9-  0004
72FB-  080E (LIT)
72FD-  0003
72FF-  0A47 PICK
7301-  080E (LIT)
7303-  0002
7305-  0A47 PICK
7307-  080E (LIT)
7309-  0003
730B-  0A6C ROLL
730D-  08E7 DROP
730F-  080E (LIT)
7311-  0004
7313-  0A6C ROLL
7315-  08E7 DROP
7317-  080E (LIT)
7319-  0012
731B-  7277 XYSWAP
731D-  080E (LIT)
731F-  0034
7321-  7277 XYSWAP
7323-  41DC .
7325-  41DC .
7327-  41DC .
7329-  41DC .
732B-  0DA3 ;S
732D-  ---- --
```

**Figure One**

```
SCR# 53
( TESTING OF THE WORDS DUPx, DROPx, AND SWAPxy   840415 S.CHEN )

  CR         1 2 3 4
  DUP3     ( 1 2 3 4 2    )      DUP2    ( 1 2 3 4 2 4 )
  DROP3    ( 1 2 3 2 4    )      DROP4   ( 1 3 2 4     )
  SWAP12   ( 1 3 4 2      )      SWAP34  ( 3 1 4 2     )
  . . . .  ( THE FINAL PRINT OUT SHOULD BE:   2 4 1 3 )

: Z
  CR         1 2 3 4
  DUP3     ( 1 2 3 4 2    )      DUP2    ( 1 2 3 4 2 4 )
  DROP3    ( 1 2 3 2 4    )      DROP4   ( 1 3 2 4     )
  SWAP12   ( 1 3 4 2      )      SWAP34  ( 3 1 4 2     )
  . . . .  ( THE FINAL PRINT OUT SHOULD BE:   2 4 1 3 )
; Z

SCR# 54
( WORDS USED TO DEFINE DUPx, DROPx, AND SWAPxy   840415 S.CHEN )

79-STANDARD:

   DUP DROP SWAP    OVER ROT PICK ROLL    >R R> R@

   + 2+ /MOD ! @     IF THEN     BASE CONVERT   :  ;

   IMMEDIATE LITERAL  STATE    COMPILE   [COMPILE]

   ( HERE


NON STANDARD (BUT NOT QUITE UNKNOWN):

   SP@  WIDTH
SCR# 55
( IN-WORD PARAMETER PASSING: DUPx DROPx SWAPxy   840415 S.CHEN )

: INWORD$ BASE @ >R 16 BASE ! HERE + 0 0 ROT CONVERT DROP DROP
  [COMPILE] LITERAL R> BASE ! ;

: ?COMPILE STATE @ IF R> DUP @ , 2+ >R THEN ;

: XYSWAP 16 /MOD >R DUP + SP@ + SP@ R> DUP + + OVER @ OVER @
  >R SWAP ! R> SWAP ! ;

 3 WIDTH ! : DUPX   3 INWORD$ ?COMPILE PICK   ; IMMEDIATE

 4 WIDTH ! : DROPX  4 INWORD$ ?COMPILE ROLL
                              ?COMPILE DROP   ; IMMEDIATE
           : SWAPXY 4 INWORD$ ?COMPILE XYSWAP ; IMMEDIATE
31 WIDTH !

[1] : INWORD$ BASE @ >R 16 BASE !
      HERE + 0 0 ROT CONVERT DROP  DROP  ?DUP 0=
      IF ." INVALID HEX IN WORD" ABORT THEN
      [COMPILE] LITERAL R> BASE ! ;



[2] : INWORD$ BASE @ >R 16 BASE !   DUP >R
      HERE + 0 0 ROT CONVERT C@ DUP 0= SWAP BL = OR NOT
      IF HERE COUNT TYPE ."  ? INVALID HEX " HERE
         COUNT R@ - SWAP  R> + SWAP  TYPE  ABORT
      THEN  R> DROP DROP
      [COMPILE] LITERAL R> BASE ! ;
```

**Figure Two**

defined in fig-FORTH and most other Forth systems. For those unfamiliar with these words, please refer to *Forth Encyclopedia* by Derrick and Baker.

The key point to our improvement lies in the special words **INWORD$** and **?COMPILE**. The first word decodes the in-word parameter as a hexadecimal number and then **LITERAL**s it. Its input is the **WIDTH** declared prior to the colon definition. The second word conditionally **COMPILE**s the word next to it when its calling word is executed in compiling mode; otherwise it does nothing.

With the support of these two special tools, the definitions of **DUPx**, **DROPx** and **SWAPxy** become quite simple, as detailed in screen 55. In the definition of **SWAPxy**, one will find a word **XYSWAP**. It is predefined to swap two stack items specified by two hex digits juxtaposed in the form of a number. For example, the entry

**HEX 15 XYSWAP**

will swap the top item and the fifth item on the stack.

Our improvement and testing have been done on an Apple II with MicroMotion Forth-79, version 2.0. The test word **Z**, mentioned above, now works perfectly. Its actual object code is shown in Figure One, along with the result of its execution. In addition, the word **INWORD$** could be implemented with some error checking like that in Figure Two.

# ATTENTION FORTH AUTHORS!

## Author Recognition Program

To recognize and reward authors of Forth-related articles, the Forth Interest Group adopted the following Author Recognition Program, effective October 1, 1984.

Articles

The author of any Forth-related article published in a periodical or in the proceedings of a non-Forth conference is awarded one year's membership in the Forth Interest Group, subject to these conditions:

a. The membership awarded is for the membership year following the one during which the article was published.

b. Only one membership per person is awarded in any year, regardless of the number of articles the person published in that year.

c. The article's length must be one page or more in the magazine in which it was published.

d. The author must submit the printed article (photocopies are accepted) to the Forth Interest Group, including identification of the magazine and issue in which it appeared, within sixty days of publication. In return, the author will be sent a coupon good for the following year's membership.

e. If the original article was published in a language other than English, the article must be accompanied by an English translation.

f. Articles are eligible under this program only if they were first published after October 1, 1984.

Letters to the Editor

Letters to the editor are, in effect, "mini-articles," and so deserve recognition. The author of any Forth-related letter to an editor published in any magazine *except Forth Dimensions*, is awarded $10 credit toward FIG membership fees, subject to these conditions:

a. The credit applies only to membership fees for the membership year following the one in which the letter was published.

b. The maximum award in any year to any person will not exceed the full cost of the membership fee for the following year.

c. The author must submit to the Forth Interest Group a photocopy of the printed letter, including identification of the magazine and issue in which it appeared, within sixty days of publication. The author will then be sent a coupon worth $10 toward the following year's membership.

d. If the original letter was published in a language other than English, the letter must be accompanied by an English translation.

e. Letters are eligible under this program only if they were first published after October 1, 1984.

# TI 99/4A Screen Dump

*Howard H. Rogers*
*Torrance, California*

Forth, as written for the Texas Instruments 99/4A computer, permits a simple implementation of bit-map mode. The Forth program described below prints a full-size replica of the screen on a compatible printer. This technique should be adaptable to other computers, if changes make the scanning procedures specific to the particular video display processor (VDP). The printer used was a Star-Micronics Gemini 10. The resolution attained is identical to that of the screen, 256 x 192 pixels.

The program operates in the following way. The particular bit-map mode, as stored in **VDPMDE** (the word for VDP control; screen 123, line 9), selects the addresses which determine the limits of the specific mode in use. VDP RAM (Figure One) is scanned (screen 123, lines 2-5) to correspond to the first row of pixels starting at the upper-left corner of the screen. The data is stored in the variable **SCRDATA** until the line is complete (screen 123, line 5).

At this point, the method used by **PRT** (screen 122, lines 13-15) to achieve the desired resolution in the final printout will be described. The bit-map modes available in the printer have too many dots per inch to be useful (sixty or 120), since a print would result which is much smaller than the image on the ten-inch monitor screen. The greatest resolution available in character mode was seventeen characters per inch, which was not enough (136 characters total in eight inches). The problem was solved by selecting three characters, ASCII 225, 227 and 231, which print as follows. An ASCII 225 prints a 3 x 3 square in the upper-left corner of a 6 x 6 field; ASCII 227 prints a 3 x 3 square in the upper-right corner of the field; and ASCII 231 prints a 3 x 6 rectangle in the top half of the field. Hence, a maximum of 272 3 x 3 squares in eight inches resulted. A linefeed of 2/72 inch gives the proper ratio between height and width, and also provides some

| ROWS | COL. 0-7 | COL. 8-F | COL. 10-17 | COL. 18-1F | | COL. A0-A8 | | COL. F8-FF |
|---|---|---|---|---|---|---|---|---|
| 0 | 2000 | 2008 | 2010 | 2018 | | 20A0 | | 20F8 |
| 1 | 2001 | 2009 | 2011 | 2019 | | 20A1 | | 20F9 |
| 2 | 2002 | 200A | 2012 | 201A | | 20A2 | | 20FA |
| 3 | 2003 | 200B | 2013 | 201B | | 20A3 | | 20FB |
| . | . | . | . | . | . . . . | . | . . . | . |
| 7 | 2007 | 200F | 2017 | 201F | | 20A7 | | 20FF |
| 8 | 2100 | 2108 | 2110 | 2118 | | 21A0 | | 21F8 |
| . | . | . | . | . | . . . . | . | . . . | . |
| F | 2107 | 210F | 2117 | 211F | | 21A7 | | 21FF |
| . | . | . | . | . | . . . . | . | . . . | . |
| B0 | 3600 | 3608 | 3610 | 3618 | | 36A0 | | 36F8 |
| . | . | . | . | . | . . . . | . | . . . | . |
| BF | 3707 | 370F | 3717 | 371F | | 37A7 | | 37FF |

**Partial Memory Map of Screen in Bit-Map Mode.**

**Figure One**

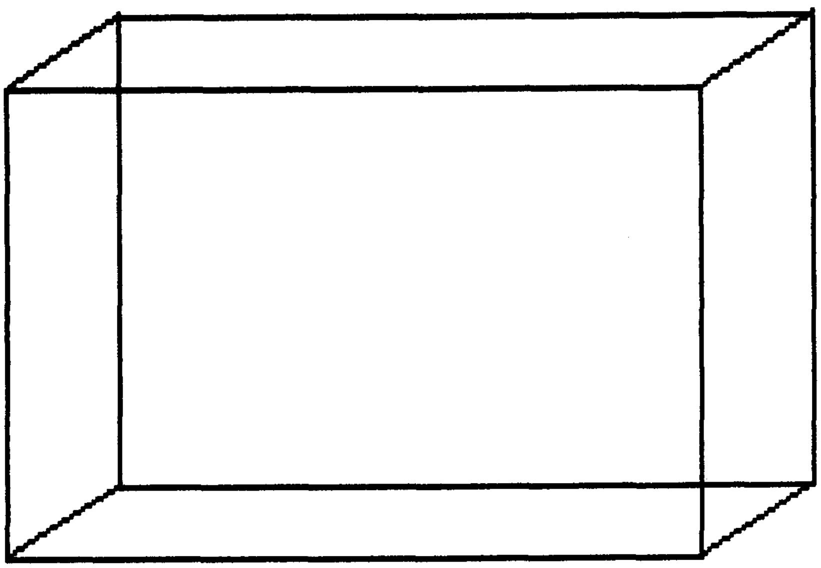
```
SCR #122
 0 ( SCREEN DUMP - BIT-MAP MODES   HHR  1984 ) BASE->R   DECIMAL
 1 0 VARIABLE SCRDATA 256 ALLOT SCRDATA 256 ERASE ( VDP data )
 2 0 VARIABLE STDATA  128 ALLOT STDATA  128 ERASE ( Printer data )
 3 SWCH 15 EMIT 27 EMIT 65 EMIT 2 EMIT  ( Printer codes  GEMINI 10)
 4  27 EMIT 77 EMIT 4 EMIT UNSWCH ( 2/72 LF, 4 LH margin, 17 CPI)
 5 : SELECT CASE  0 OF 32 ENDOF ( Choose printer character)
 6           128 OF 225 ENDOF    ( Binary-ASCII    0- 32 01-227)
 7            64 OF 227 ENDOF    ( Binary-ASCII   10-225 11-231)
 8           192 OF 231 ENDOF  ENDCASE ;
 9 : SCAN 32 0 DO SCRDATA I + C@  ( Read data, 32 bytes per Line)
10     4 0 DO DUP 192 AND SELECT ( Select 2 bits for each character)
11    STDATA J 4 * I + + C!       ( Store data to be sent to printer)
12    2 SLA LOOP DROP LOOP  ;     ( Shift left 2 bits each loop)
13 : PRT SCAN SWCH 128 0 DO STDATA I + ( Begin print & fetch data)
14     C@   EMIT8 LOOP CR         ( Send CHR 32, 225, 227, and 231)
15     20000 0 DO LOOP UNSWCH ;  -->  ( Time delay for printer)

SCR #123
 0 ( SCREEN DUMP - BIT-MAP MODES   HHR 1984   Continued)
 1
 2 : SCREEN_DUMP  DO 8 0 DO 256 0 DO  ( Reads VDP RAM and  )
 3    R> R> R> R>  R  SWAP >R OVER +  ( stores in CPU RAM  )
 4     SWAP >R SWAP >R OVER + SWAP >R ( Adds 3 loop indicies)
 5    SCRDATA I 8 / + 8 VMBR 8 +LOOP  ( Transfer from VDP to CPU)
 6    PRT ?TERMINAL IF SWCH CR UNSWCH
 7    QUIT ENDIF  LOOP              ( Press CLEAR to stop printing  )
 8    256 +LOOP BEEP ;              ( Signals completion of printing)
 9 : SCREEN/DUMP  VDPMDE @   CASE
10        4 OF   14336 8192 ENDOF  ( Selects Graphics2 mode )
11        5 OF   12288 8192 ENDOF  ( Selects SPLIT mode      )
12        6 OF   14336 9216 ENDOF  ( Selects SPLIT2 mode     )
13     HONK CR ." ERROR- REQUIRES BIT-MAPPED MODES " QUIT ENDCASE
14    SCREEN_DUMP  BEEP ; R->BASE ( If none of above modes )
15
```

overlap of characters. The printer codes are shown on screen 122, lines 3-4.

**PRT** starts with do loops that read data for each line of the screen. Each two bits of data are stored sequentially in **STDATA** by **SCAN** (screen 122, lines 9-12). Upon completion of the loop, printing begins with **SWCH**, which turns on the interface to the printer. After the line has been printed by the print loop (screen 122, lines 13-15), the process continues line-by-line, using the three do loops in screen 123, lines 2-8, until the entire screen has been copied.

A sample use of the graphics program (screen 132) and the printed result are in Figure Two, a simple three-dimensional box. The lines were first plotted on the monitor screen with the word **.LINE** (available on the TI 99/4A) and then were printed with **SCREEN/DUMP**. The last step usually requires about ten minutes, partly due to the unidirectional print mode used for the greatest precision when plotting graphical data.

```
SCR #132
  ( EXAMPLE OF BIT MAP MODE DRAWING)
  ( RECTANGULAR BOX)

   GRAPHICS2 DRAW
    20 171 205 171 LINE     205 171 205  40 LINE
   205  40  20  40 LINE      20  40  20 171 LINE
   205 171 235 151 LINE     205  40 235  20 LINE
    20  40  50  20 LINE      50  20 235  20 LINE
   235  20 235 151 LINE      50 151 235 151 LINE
    50 151  50  20 LINE      20 171  50 151 LINE
```



## An Interactive Math Game:

# Mathquiz

*Lyle D. Morton*
*Wheatland, Wyoming*

Perhaps you work from the same handicap I do. I am not an expert at Forth. With that in mind, I am going to present an article for all the Forth beginners in the world. When I wrote Mathquiz, I accomplished many routines and structures for the first time in a program of my own design. Some of the words and concepts are straight out of *Starting Forth* by Leo Brodie, but in this program I tried to put them together in a meaningful way.

As the father of two computer-loving children, I was seeking to develop something they would enjoy using, and which could also help with their basic math skills. I wrote it in Forth as a self-teaching exercise, and because I prefer it to BASIC and Pascal. In an attempt to emulate such hardware as Texas Instruments' "Little Professor" and "Speak and Math," I included the skill-level selection routines and congratulatory remarks. The title page and the pep talk at the start of the program were an attempt to spruce up the program and give it the feel of a software package the kids might be exposed to in school.

Mathquiz is written in MVP-FORTH (Forth-79), but I have added a couple of extra Apple utilities I coded on my Apple II. Those utilities are **INVERSE** and **NORMAL**. They make use of inverse characters on the screen and can easily be deleted from the program. I have included their definitions for Apple users who have not yet coded them. You will also notice that I used a large number of variables in the program. I have read that this shows up in most programs written by beginners, but I think it enhances readability of the code.

If you have a machine that uses eighty columns exclusively, you will want to change the title page to use the full width, or to include carriage returns after line endings, to eliminate scrambled eggs for a title. You may also want to adjust the delay loops to give the proper amount of time to read

```
   HELLO.  YOU ARE ENTERING THE WORLD
OF MATH-QUIZ.  PLEASE ENJOY YOURSELF!
BE ADVISED THE UPPER LEVELS MAY REQUIRE
A PENCIL AND PAPER.  GOOD LUCK! WELCOME TO MATHQUIZ VERSION 1.3 !
ENTER YOUR FIRST NAME.
LYLE
LYLE , YOU MAY CHOOSE FROM THE
FOLLOWING MENU :

     MATH QUIZ VERSION 1.3
        1.  ADDITION
        2.  SUBTRACTION
        3.  MULTIPLICATION
        4.  DIVISION
        5.  STOP PROGRAM
TYPE THE NUMBER OF YOUR CHOICE.


CHOSE YOUR LEVEL OF DIFFICULTY:
     1, 2, 3, OR 4

9   + 5   = 14 CORRECT
10   + 2   = 12 CORRECT
3   + 9   = 12 CORRECT
5   + 0   = 5 CORRECT
1   + 1   = 2 CORRECT
1   + 7   = 8 CORRECT
10   + 10   = 20 CORRECT
3   + 7   = 10 CORRECT
10   + 9   = 19 CORRECT
9   + 0   = 9 CORRECT
LYLE, YOU WERE CORRECT 10 TIMES
OUT OF 10.  VERY GOOD!LYLE , YOU MAY CHOOSE FROM THE
FOLLOWING MENU :

     MATH QUIZ VERSION 1.3
        1.  ADDITION
        2.  SUBTRACTION
        3.  MULTIPLICATION
        4.  DIVISION
        5.  STOP PROGRAM
TYPE THE NUMBER OF YOUR CHOICE.


CHOSE YOUR LEVEL OF DIFFICULTY:
     1, 2, 3, OR 4

28   - 2   = 26   CORRECT
34   - 18   = 16   CORRECT
40   --17   = 13
WRONG, THE CORRECT ANSWER IS 23
38   - 11   = 27   CORRECT
28   - 11   = 17   CORRECT
40   - 14   = 26   CORRECT
36   - 0   = 36   CORRECT
34   - 5   = 29   CORRECT
29   - 6   = 23   CORRECT
39   - 6   = 33   CORRECT
LYLE, YOU WERE CORRECT 9 TIMES
OUT OF 10.  VERY GOOD!LYLE ,
YOU MAY CHOOSE FROM THE
FOLLOWING MENU :

     MATH QUIZ VERSION 1.3
        1.  ADDITION
        2.  SUBTRACTION
        3.  MULTIPLICATION
        4.  DIVISION
        5.  STOP PROGRAM
TYPE THE NUMBER OF YOUR CHOICE.

CHOSE YOUR LEVEL OF DIFFICULTY:
     1, 2, 3, OR 4
20 * 14 = 280 CORRECT!
19 * 5 = 95 CORRECT!
5 * 12 = 60 CORRECT!
3 * 16 = 58
WRONG, THE CORRECT ANSWER IS 48
6 * 18 = 108 CORRECT!
6 * 15 = 90 CORRECT!
10 * 10 = 100 CORRECT!
6 * 5 = 30 CORRECT!
7 * 15 = 115
WRONG, THE CORRECT ANSWER IS 105
8 * 16 = 118
WRONG, THE CORRECT ANSWER IS 128
LYLE, YOU WERE CORRECT 7 TIMES
OUT OF 10.  NOT BAD!LYLE ,
YOU MAY CHOOSE FROM THE
FOLLOWING MENU :
     MATH QUIZ VERSION 1.3
        1.  ADDITION
        2.  SUBTRACTION
        3.  MULTIPLICATION
        4.  DIVISION
        5.  STOP PROGRAM
TYPE THE NUMBER OF YOUR CHOICE.
THANKS-PLAY AGAIN SOON! OK
```

**Figure One**

the title page and pep talk. If you are memory conscious, you can use fewer than forty bytes in **NAME** (most names are not nearly that long).

All the math routines were straightforward except division. Since we normally don't have access to floating point in Forth, and not wanting answers truncated, I decided to use a modified multiplication routine. I multiply two random numbers, then display one of them and the product, and ask the player to figure out the second multiplier. (By adding one to the original random numbers, I eliminated the possibility of dividing by zero.)

Those of you who are as green as I was when I started this program may want to look at **GETNAME** and **ANSWER**. These were the two most difficult words for me to get correct. If there is a lack in *Starting Forth*, I think this is it. It took several hours of research in *All About Forth* by Glen Haydon and in *And So FORTH* by Timothy Huang before I knew what I was doing with this kind of input in an interactive program.

Mathquiz has been in transition for many months. Hence, this is version 1.3. Originally, the program did not loop continuously; after the initial **START** command was typed by the player, only one set of ten problems was given. At the end of those problems, the player needed to type **QUIZ** to get his next ten problems. I made the program loop by using a **BEGIN WHILE REPEAT** structure in the word **QUIZ**. Recently, I changed the method of problem type and difficulty selection to single-keystroke entry by using **KEY** and by comparing the ASCII value it returns for a match.

I have several more ideas to improve (maybe "change" is the correct word) Mathquiz. Changes often have meant incorporating different Forth structures, and this has been both enjoyable and enlightening. I hope the ideas in Mathquiz are of use to you.

```
SCR #71
( RANDOM NUMBER GENERATOR              from "Starting FORTH"  )
VARIABLE RND                           HERE RND !
: RANDOM   RND @ 31421 * 6927 + DUP RND ! ;

: CHOOSE   ( U1---U2 )
     RANDOM U* SWAP DROP ;

( APPLE UTILITIES   INVERSE & NORMAL                09/83 LDM )

HEX
: INVERSE   FE80 CALL ; ( Change to inverse characters )
: NORMAL    FE84 CALL ; ( Return to normal characters )
DECIMAL



SCR #79
( MENU MODULE )
: BEGINLINE   CR 8 SPACES ;
: HEADER   CR 5 SPACES
     ." MATH QUIZ VERSION 1.3 " ;
: MATHMENU   ( DISPLAY CHOICES )
    HEADER BEGINLINE ." 1. ADDITION"
    BEGINLINE ." 2. SUBTRACTION "
    BEGINLINE ." 3. MULTIPLICATION"
    BEGINLINE ." 4. DIVISION "
    BEGINLINE ." 5. STOP PROGRAM" CR
    ." TYPE THE NUMBER OF YOUR "
    ." CHOICE. " KEY CHOICE ! ;




SCR #80
( DIFFICULTY SELECTION MODULE )
: DIFFICULTY   CR ." CHOSE YOUR LEVEL"
    ."  OF DIFFICULTY:" CR 5 SPACES
    ." 1, 2, 3, OR 4  " ;
: SELECT    BEGIN KEY DUP 2DUP    ( use BEGIN ... UNTIL loop )
    49 = IF 11 LEVEL ! DROP 2DROP 1 ELSE   ( to check for err )
    50 = IF 21 LEVEL ! 2DROP 1 ELSE    ( 11,21,51,101 are )
    51 = IF  51 LEVEL ! DROP 1 ELSE    ( difficulty  )
    52 = IF  101 LEVEL ! 1 ELSE CR    ( multipliers )
    ." NUMBER OUT OF RANGE! TRY "
    ." AGAIN. " 7 EMIT ( BELL ) 0
    THEN THEN THEN THEN 1 = UNTIL ;

: GETDIF   DIFFICULTY SELECT ;


SCR #83
( SELECTION MODULE )
: GETONE   CHOICE @ DUP DUP 2DUP ( select problem type )
    49 = IF 2DROP 2DROP GETDIF 10-ADD ELSE
    50 = IF 2DROP DROP GETDIF 10-SUB ELSE
    51 = IF 2DROP GETDIF 10-MULT ELSE
    52 = IF DROP GETDIF 10-DIV ELSE
    53 = IF CR ." THANKS-PLAY AGAIN SOON!" CR
    ELSE CR CR 7 EMIT ( bell ) ." ENTRY OUT OF RANGE "
    1 CHOICE ! ( false response to cause QUIZ to loop )
    THEN THEN THEN THEN THEN ;
: QUIZ   BEGIN PAGE     ( infinite BEGIN..WHILE..REPEAT loop )
    REGREET MATHMENU   ( get player's name & show menu )
    GETONE CHOICE @    ( get player's choice )
    53 < WHILE DELAY REPEAT ;  ( check for stop or error )
: START   MATHTITLE DELAY INSTRUCT 2DELAY
    2DELAY COMMENCE 2DELAY QUIZ ;
```

Now You Can Add

New!

# ARTIFICIAL INTELLIGENCE

To Your Programs Using a Powerful Combination

"SUPERFORTH 64" +AI

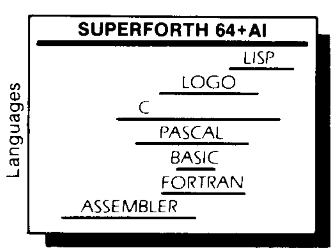
By Elliot Schneider & Jack Park

## Heres Your Chance to Profit by being on the Forefront, Write 5th Generation Software

**Learn How To:**
- Create Intelligent Programs
- Build Expert Systems
- Write Stand Alone License Free Programs
- Construct Rule Bases
- Do Knowledge Engineering
- Use Inference Engines

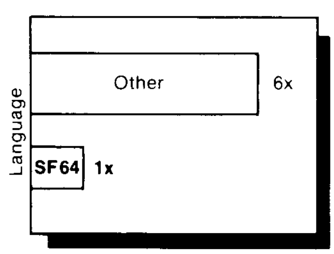
**Write Intelligent Programs For:**
- Home Use
- Robotics
- Medical Diagnosis
- Education
- Intelligent CAI
- Scientific Analysis
- Data Acquisition
- Data Analysis
- Business
- Real Time Process Control
- Fast Games
- Graphics
- Financial Decisions

**Extended Math Functions**
- Fast ML Floating Point & Integer Math
- Double Precision 2E+38 with Auto. Sci Not.
- $n^x e^x$ Logx Loge Sin Cos Tan SQR 1/X...
- Matrix and Multidimensional Lattice Math
- Algebraic Expression Evaluator




Power of Languages Constructs

SuperForth 64 is more powerful than most other computer languages




Programming Time

SuperForth 64 Saves You Time and Money

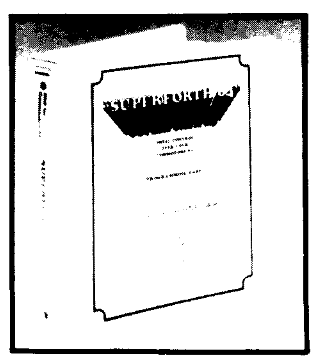
**Easy Graphics & Sound Words**
- Hires Plotting
- Windows
- Split Screen
- Printer/Plotter Ctrl
- Sprite & Animation Editor
- Turtle Graphics
- Koala Pad Graphics Integrator
- Hires Circle, Line, Arc
- Music Editor
- Sound Control

**Easy Control of all I/O...**
- RS232 Functions
- Access all C-64 Peripherals
- Interrupt Routines

**Utilities**
- Interactive Interpreter
- Forth Virtual Memory
- Full Cursor Screen Editor
- Full String Handling
- Trace & Decompiler
- Conditional Macro Assembler
- Interactive Compiler
- Romable Code Generator
- 40K User Memory
- All Commodore File Types
- Conversational User Defined Commands

**Great Documentation**
- Easy to Read 350 pg. Manual with Tutorials
- Source Screen Provided
- Meets all MVP Forth-79 Industrial Standards
- Personal User Support

**A Total Integrated Package for the Commodore 64**



**Ordering Information:** Check, Money Order (payable to MOUNTAIN VIEW PRESS, INC.), VISA, MasterCard, American Express. COD's $5.00 extra. 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 $10.

SPECIAL INTRODUCTORY OFFER

only $99.00

203.00 Value
Limited Time Offer

**Call:**
**(415) 961-4103**
**MOUNTAIN VIEW PRESS INC**
P.O. Box 4656
Mt. View, CA 94040

Dealer for
**PARSEC RESEARCH**
Drawer 1776, Fremont, CA 94538

# Local Definitions

*Carol Pruitt*
*Rochester, New York*

As I've developed larger and larger systems in Forth, I've become increasingly annoyed by its habit of remembering every name. Most utility packages consist of dozens of words, of which only a handful (scattered here and there) are intended for outside use. From the compiler's point of view, the rest of those words are just names that can't be used for something else. Mere assemblers can distinguish between local and global definitions — why shouldn't Forth?

I've tried using vocabularies for this purpose, but have become weary of constantly invoking this vocabulary and that. (The readability of the code suffers, too.) After six years of Forth programming, I've become weary enough to do something about it.

The most visible word (see Figure One) in my implementation is **LOCAL**, which is used like **IMMEDIATE**. It marks the most recently defined word as a local definition. **LOCAL.START** and **LOCAL.END** determine the boundaries of the local area (i.e., the area inside which the local definitions are visible).

This technique serves the intended purpose, namely:

1) Only generally-useful words are visible outside their area.

If this is rephrased as, "Forth can tell the difference between local and global words," a corollary advantage becomes apparent:

2) Humans reading the code can tell which words are intended for general use.

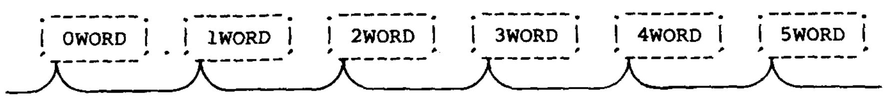
If, on the other hand, the phrasing is, "It is impossible to call local words from the keyboard," then:

3) This technique can serve much the same purpose as sealed vocabularies.

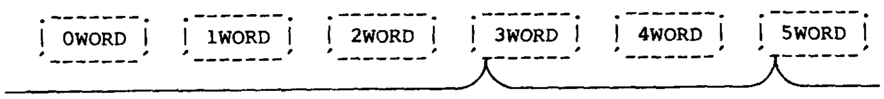
A fourth advantage derives from the implementation. My first inclination was simply to smudge all local words upon reaching the end of the local area. However, the technique shown here, that of delinking the local words

```
( Example of LOCAL usage                                11-Jun-84 CJP )

LOCAL.START                                    ( begin local area     )

0 VARIABLE OWORD                       LOCAL   ( note that            )
                                               (    local definitions)
CODE 1WORD ( whatever ) NEXT,          LOCAL   (    can be variables,)
                                               (    code words,       )
: 2WORD     ( whatever ) ;             LOCAL   (    colon words, etc.)

: 3WORD     1WORD 2WORD ;                      ( note also that       )
                                               (    locals & globals )
: 4WORD     10 0                               (    can be            )
            DO 1WORD LOOP ;            LOCAL   (    intermixed        )
                                               (    as desired        )
: 5WORD     OWORD
            IF 3WORD ELSE 4WORD THEN ;

LOCAL.END                                      ( end local area       )

;S
```

**Figure One**




**Just before interpretation of LOCAL.END, all words of the example are linked into the CURRENT vocabulary, just as in any typical Forth system.**




**After interpretation of LOCAL.END, only 3WORD and 5WORD are still linked into the vocabulary. This frees the other four names for re-use, and incidentally speeds compilation.**

**Even though the four local words are no longer linked into the vocabulary, however, they are still used by other words within the local area. They can be used from outside the local area only indirectly, by using global words which use the local words.**

**Figure Two**

```
( LOCAL: Implement local definitions          11-Jun-84 CJP )

 0  VARIABLE LINK.FROM        ( save LFA of last non-local word)

64  CONSTANT LBIT             ( value of name-field's local bit)

: LOCAL                       ( <>--<>                        )
        CURRENT @ PFA @ NFA     ( set "local" bit             )
        DUP C@ LBIT OR          (    in name field of word    )
        SWAP C! ;               (    most recently defined    )

: LOCAL.START                 ( <>--<start addr of local area> )
        HERE ;

: LOCAL.END                   ( <start addr of local area>--<> )
                              ( de-link LOCAL words from vocab.)
        CURRENT @ PFA
        DUP LINK.FROM !         ( save starting link-from addr)
        @                       ( initialize this-word pointer)
        BEGIN 2DUP <            ( for each word defined       )
        WHILE                   (    since LOCAL.START,       )
           DUP NFA C@ LBIT AND
           IF LFA @             (    if the word is local,    )
              DUP LINK.FROM @ ! (       de-link it            )
           ELSE LFA             (    else it becomes          )
              DUP LINK.FROM !   (       link-from word        )
              @
           THEN
        REPEAT 2DROP ;
;S
```



**Figure Three**

from their vocabulary (see Figure Two), yields this happy side effect:

4) Things compile faster.

As a test of this side effect, I marked enough local definitions in the lower layers of a large application to reduce the total number of global words (in system and application) by twenty-three percent. Compile time for the upper layer fell by twenty-two percent. Presumably, the similarity of percentages is not a coincidence.

Of course, any local-definition technique has the following bizarre (to the veteran Forth programmer) effect: once a word has been designated as local, it cannot be called (or even found) from another part of the system (or even the keyboard) until that designation is removed and the source is recompiled. Since locals are most useful on large applications (which take a long time to recompile), care should be exercised in determining which words are to be local.

**Implementation**

LLE Forth, used in the University of Rochester's Laboratory for Laser Energetics, is quasi-79-Standard; I believe the additional words in Figure Three are familiar to *Forth Dimensions* readers. **NFA**, **LFA** and **PFA** each start with the code-field address on the stack, and leave the name-field, link-field or parameter-field address, respectively.

Our link field points to the code field of the preceding word. The definition of **LOCAL.END** can be modified to suit other types of link.

Some way must be chosen to flag local words for delinking. The count byte of our name field had two unused bits, so I chose bit six for this purpose. This required modifying **FIND** to mask out bit six along with the precedence bit. Some Forth systems may already mask out everything except the count (and smudge bit, if any).

If enough spare bits (or other flags) are available, additional levels of localness could be formalized. The implementation given here is, however, semi-nestable. That is, smaller, super-local areas can be designated inside larger local areas. The only restriction is that a local definition cannot exist inside a super-local area. (When **LOCAL** is used inside a super-local area, it designates a super-local word.)

There is also one fairly minor restriction on the use of local words in general: while it is possible to change vocabularies inside a local area, it is definitely not advisable. The effects are peculiar and, as a result, the code is less readable. This restriction is more academic than actual, however, since local words can serve as an alternative to vocabularies.

**Acknowledgments**

This work was supported by the U.S. Department of Energy Office of Inertial Fusion under contract DE-AC08-80DP40124 and by the Laser Fusion Feasibility Project at the Laboratory for Laser Energetics which has the following sponsors: Empire State Electric Energy Research Corporation, General Electric Company, New York State Energy Research and Development Authority, Northeast Utilities Service Company, Southern California Edison Company, The Standard Oil Company and University of Rochester. Such support does not imply endorsement of the content by any of the above parties.

# Enhanced DO LOOP

*Michael Hore*
*Numbulwar, Northern Territory*
*Australia*

This article will show how, with very little effort, you can add a useful enhancement to the Forth-83 **DO LOOP** construct. In the process, we will touch on various aspects of its implementation. If you have not tried adding any enhancements to your compiler's control structures as yet, this may be a good place to start. Sorry, if your Forth has the earlier type of **DO**, you won't be able to make use of this idea — it depends on how the Forth-83 **LEAVE** is usually implemented.

First, let's look at an example of a very common situation. You have to write a word to search a table for an entry matching a given value. The required stack effects are

*( value table-addr #entries — addr' )*
where *addr'* is the address of the matching entry in the table, or zero if there is no match.

Simple. You grab your pencil and write

```
: LOOKUP OVER + SWAP
  DO I @ OVER = IF DROP I LEAVE THEN
  LOOP ;
```

Of course, being a capable Forth programmer, in no time at all (certainly before testing!) you notice that this code will only work if there is a match in the table. If there isn't, after the **LOOP** falls through, you won't have the required zero on the stack; *value* will still be there. You need a **DROP 0**. But how can you prevent that from destroying the (correct) result when the search succeeds? The usual method is to push a zero just before the **LOOP**, in case this is the last time around. Remember, in Forth-83 there is no straightforward way of checking that this is, in fact, the last time around. You have to put the zero there just in case it is. So, after a few adjustments to keep the stack in order, you end up with

```
: LOOKUP 0 ROT ROT OVER + SWAP
  DO DROP I @ OVER =
  IF I LEAVE
  THEN 0 LOOP SWAP DROP ;
```

And that should work. But it is hardly elegant code! Now, a good rule of thumb is that if the solution to a problem looks messy, then maybe we haven't understood the problem. I know there may be such things as messy problems to which only messy solutions are possible, but without entering a long philosophical debate, the fact is that here is what appears to be a simple problem to which we have a rather messy-looking solution.

So, where does the messiness come from? The cause of the trouble is surely this: When the search succeeds, we can put anything we like after the **IF** to set up whatever results are required. But if the search fails and the **LOOP** falls through, there is no place to set up whatever results we need for that particular situation. The code after the **LOOP** is executed no matter what happened. We have to set up our "failure" results before the loop, to be there in case the loop falls through, and **DROP** them at the beginning. It's not hard to think of other situations which would give even messier code than we have here.

What we really need is a place where "failure" results can be set up, which are simply not executed if the loop **LEAVE**s. I am proposing the construct

**LOOP--FALLTHRU: . . . THEN**

I am using **THEN** to conclude the construct, since **LOOP--FALLTHRU:** is something like an **IF** combined with **LOOP** — testing the condition, "Did the **LOOP** fall through?"

Now, this construct is very easy to implement under most versions of Forth-83. The details, however, are dependent on how each particular version handles **DO LOOP**. In what follows, I will make particular reference to F83, the implementation by Henry Laxen and Michael Perry, which looks like it is becoming widely used. I will also provide enough information so that if you have a different implementation, you will at least have a good chance of working out what to do. Also, anything I say about **LOOP** could equally well apply to **+LOOP**, so I won't mention it explicitly.

**DO** is always immediate and compiles **(DO)**, its run-time operation, and pushes the address of the word following the **(DO)** onto the stack (using the system extension word **<MARK** or its equivalent). This address or "mark" may be used in different ways. It is generally picked up by **LOOP** to compile **(LOOP)** and a backward branch address pointing to the word following the **(DO)**. This branch address is compiled by the word **<RESOLVE** or its equivalent, within the definition of **LOOP**.

In pre-83 Forths, nothing else had to be done. In Forth-83, however, **LEAVE** must cause a branch out of the loop, and we have to arrange things so that this works properly. **LEAVE** is now immediate, and could compile **(LEAVE)** and then do a **>MARK** to compile a forward branch address which would be filled in ("resolved") by a **>RESOLVE** within **LOOP**. However, as there could be any number of **LEAVE**s (or none) within a loop, **>RESOLVE** would need to be done a variable number of times. This is quite possible but a little tricky, so other methods have been proposed.

One of these is for **LOOP** to work the same way, but for **(DO)** to carry a forward branch address to the word following **(LOOP)**. At compile time, this forward reference is resolved by a **>RESOLVE** within **LOOP**. **LEAVE** compiles **(LEAVE)** and a backward reference to the forward branch address following **(DO)**. This is the method proposed by Bill Stoddart (*Forth Dimensions* V/4). As he points out, this method allows a very simple implementation of a conditional **DO LOOP** which is skipped entirely if the initial index and limit are equal.

Another possibility also involves **(DO)** carrying a forward branch address, but at run time **(DO)** uses this address to put a third entry on the return stack, pointing to the word following **(LOOP)**. This is then used by **(LEAVE)**, which now needn't carry a branch address itself (and, in fact, **LEAVE** needn't have been immediate at all, except that it is required to be so by the Standard). This is the method used by Laxen and

Perry. It involves a minimal run-time penalty over Stoddart's method, but saves two bytes (on most machines) for every **LEAVE**. The F83 model also includes the conditional **DO** (called **?DO**).

Another method, which I have been using myself, is very similar except that the third return stack entry points to the word following the **(DO)** rather than to the one following the **(LOOP)**. This way, **(LOOP)** does not need a branch address either, but is just as fast to execute. **(LEAVE)** requires an extra step to work out where to go, but this is hardly significant (on the PDP-11, it is just one extra machine instruction).

Just to draw things together, in all these methods, **DO** leaves a "mark" on the stack at compile time, which is used by **LOOP** to do these things:

1. Compile a backward branch address following **(LOOP)**, that determines where the loop actually loops to. All methods except the last do this.

2. Resolve a forward branch address following **(DO)**. This determines where **LEAVE** actually leaves to. All methods except the first do this.

Our proposed implementation of **LOOP--FALLTHRU:** should work for any Forth-83 system which performs as in point two above. The key fact is that if the loop falls through we don't branch anywhere, so we don't have to work out where to go at all. The forward branch address (when resolved as in point two) is not used for this.

What will happen if, at **LOOP**, we don't perform that resolution right away? The fall-through situation won't be affected. But **(LEAVE)**, instead of going to the next word after **(LOOP)**, will go somewhere else — in fact, it will go to wherever we do perform the resolution. As mentioned earlier, our proposed construct is a bit like an **IF**. As it happens, **IF** also leaves a mark on the stack for later resolution. So, basically, all we have to do is make **LOOP--FALLTHRU:** "trick" the compiler into thinking an **IF** has been encountered. This is normally very easy.

Screen 100 gives the relevant definitions as they exist in F83. **?>MARK**, etc. are versions of the standard **>MARK**, etc. words, incorporating some error checking; namely, the pushing of a -1 under the mark on the stack. The corresponding resolving words **>?RESOLVE**, etc., check that the -1 is there.

Now notice that **LOOP** resolves twice, although **DO** only leaves one mark. The phrase **2DUP 2+** in the definition of **LOOP** really has the same result as if **DO** had **?>MARK ?<MARK**, which better makes clear what is being done. Logically, two separate marks are being resolved, one for a forward branch address following **(DO)** which allows **(LEAVE)** to work, and one for a backward branch address following **(LOOP)** so that it knows where to loop to.

The definition of our new word is given on screen 101. Note that it is the same as **LOOP** with the **?>RESOLVE** left out. The **?<RESOLVE** must be there, as it compiles the address used by **(LOOP)**.

That is all there is to it. When **THEN** comes along, the mark that we didn't resolve at the end of the loop is resolved by **?>RESOLVE** as though nothing unusual were happening. If **(LEAVE)** is later executed, it will leave to this point.

As the F83 error checking is identical for all control structures, nothing special need be done about the -1 flag. If your implementation uses more elaborate error checking, however, with a different flag value on the stack for each construct (as in fig-FORTH), **LOOP--FALLTHRU:** will have to alter the flag value to the correct one for **IF**.

A couple of extra points:

1. **LOOP--FALLTHRU:** can be followed by an **ELSE** as well as by a **THEN**. The **ELSE** phrase will be executed if **LEAVE** is executed in the loop. This may be useful if a loop has a number of **LEAVE**s which all require some common action to be carried out.

2. F83 includes the conditional **?DO LOOP**, as mentioned above. The address used by **(?DO)** to skip the loop is the same used by **(LEAVE)**. Thus, if **LOOP--FALLTHRU:** is used with **?DO** and the loop is skipped, it will look as if a **LEAVE** has been taken. This shouldn't be a problem, but should be kept in mind.

```
Screen 100

( Some control structures as per Laxen/Perry F83      Jul84MRH )

: IF      COMPILE ?BRANCH   ?>MARK                     ; IMMEDIATE
: THEN    ?>RESOLVE                                     ; IMMEDIATE

: DO      COMPILE (DO)      ?>MARK                     ; IMMEDIATE
: LOOP
    COMPILE (LOOP)    2DUP 2+ ?<RESOLVE ?>RESOLVE      ; IMMEDIATE
: +LOOP
    COMPILE (+LOOP)   2DUP 2+ ?<RESOLVE ?>RESOLVE      ; IMMEDIATE






Screen 101

( LOOP--FALLTHRU: for Laxen/Perry F83                 Jul84MRH )

: LOOP--FALLTHRU:
    COMPILE (LOOP)    2DUP 2+   ?<RESOLVE              ; IMMEDIATE

: +LOOP--FALLTHRU:
    COMPILE (+LOOP)   2DUP 2+   ?<RESOLVE              ; IMMEDIATE

( Example of use: )

: LOOKUP  ( value table-addr #entries -- addr' | 0 )
    OVER + SWAP
    DO    I @  OVER =    IF   DROP  I  LEAVE    THEN
    LOOP--FALLTHRU:    DROP 0    THEN       ;

CREATE TABLE   1 , 2 , 3 , 4 , 5 ,


100 LOAD 101 LOAD
 ok
TABLE .
21212  ok
1 TABLE 5  LOOKUP .
21212  ok
3 TABLE 5  LOOKUP .
21216  ok
9 TABLE 5  LOOKUP .
0  ok
```

# The Far Right Stuff

*Mike Perry, et al.*
*Berkeley, California*

There are those among us who are attempting to advance by only one generation of computer technology. It is just such an egregious lack of imagination which has put us into the dire straits in which we find ourselves today. Why stop with the fifth generation, when logic has already taken us as far as it can? It is time to throw off the Markov chains which have bound our thinking for so long. Let us retain our integrity and not sink into a morass of floating points.

In short, I propose that work begin on a sixth generation computer. When the knowledge-based expert systems of the fifth generation have exhausted the possibilities of mere rational thought, all eyes will turn to the potential of the irrational. It is not too early to develop intuitive machines, which I like to think of as ignorance-based assumption systems. They will, of course, use irrational logic gates, which are based (naturally) on the natural number base (2.7 et cetera). Ordinary logical reasoning can be compared to a chain: an inference is only as strong as its weakest link. Intuitive reasoning is more like a rope: no one strand is very strong, but when tightly coupled into a rope, conclusions are bound to be inescapable. The possibilities are imponderable.

There are many advantages to sixth generation machines. For one, ignorance systems will consume very little power: as everyone knows, Knowledge is Power. They can be built from some well-known illogic gates, such as the ignorance gate (don't confuse me with the facts), the obvious gate (as any fool can clearly see) and the sure-thing gate (it's in the mail). As all of these are output-only devices, no input will be required, greatly simplifying system design. This will lead to replacement of the old computer adage "Garbage In, Garbage Out" by simply "Garbage Out."

Performance measurement will also be affected. Instead of MIPS (millions of instructions per second) or KLIPS (kilo logical inferences per second), the sixth generation machines will be measured in DUPES (deca unreasonable predictions each second).

New languages will be required. As demonstrated below, the older languages were very limited:

ASCRAMBLER The first languages were an improvement over their predecessors, and helped to enhance problems to truly macroscopic proportions.

FORTUITOUS Filled an array of real needs at first, but it was beginner's luck. Neither its backus nor anyone else would recognize it now.

GOBBLE Government Organized Big Business LanguageE was a real turkey, too wordy and difficult to pronounce. Its users were renowned for their weight-lifting abilities.

GARGOYLE BEGIN GARGOYLE := ACADEMIC. It had a Hoareable appearance, AND was cursed time and again. It was super-setted by RASCAL in the END.

SIMPLE 10 Basically too complex and hard to use. 20 Abused the powerful COME-FROM construct.

LISTLESS (and (used Polish notation) (put old parentheses on a garbage heap)).

RASCAL While not without its Wirth, it was unforgiving and not really modular, too.

FROTH At first, even its starry-eyed followers thought it would not float, but when it came un-Moored they could see that it did. On the stack it pushed more than once has been.

GOSSIP A classy language which supported chatting in both back-fence and window modes. Some objected to passing this sort of messages, feeling that some facts were better kept private.

SEA {often confused with OCEAN; it is actually rather uniques;} SEA was one inspiration for the saying, "I'm elegant, you're terse, he's unreadable."

MONOLITH To make RASCAL suitable for numerous tasks, new types of checking were implemented for a program's definitions. A recent export of great import.

ADORD The official language of the Department of Redundancy Department, who decided to Boole their resources. Before it is valuable, it will be dated.

PROLONGED If the inferences of some experts are proved correct, then its deductions may eventually lead the way toward superficial intelligence.

OXYMORON Allows an Ocean of disputer processors to run the RISC of producing abysmal results. Sort of a contradiction in teams.

To solve the problems caused by sixth generation machines, new languages will have to be created. Here are a few possibilities:

GUESSWORK Can be used to implement functions like What-Did-I-Mean?

ABSURD Uses backward inference to reduce the solution until only the problem remains. Given a hyper-theoretical situation, it expands the problem until it interferes with the solution, thus disproving the result.

As grand as these schemes may sound, let us not think that the sixth generation is in any way the ultimate. Already the characteristics of the seventh generation are becoming apparent. Taking the chance of combining the best features of irrational logic with the technology of virtual machines, we will at last see real imaginary computers. This will lead to the replacement of the old saying, "Never trust a computer you can't lift" by the new adage, "Never trust a computer big enough to see."

# 1985 Rochester Forth Conference

June 12–15, 1985 University of Rochester Rochester, New York

**The fifth Rochester Forth Conference** is sponsored by the Institute for Applied Forth Research, Inc. in cooperation with the Laboratory for Laser Energetics of the College of Engineering at the University of Rochester, the Rochester Section of the IEEE, and the Rochester Chapter of the IEEE Computer Society.

**The focus of the Conference** is on Software Management and Software Engineering. The invited speakers will discuss the use of Forth in large projects, modifications to the language aimed at improving productivity, and contrast its use in business and scientific environments. In addition, presented papers will cover many aspects of implementing and applying Forth, including robotics, graphics, Forth chips and real-time systems. Forth novices, programmers, implementors, and project managers will find these presentations useful and pertinent to their work.

**The final day of the Conference** will be open to the public, and devoted to tutorials, demonstrations and presentations by Forth vendors, panel discussions, and poster presentations. All those interested in learning about Forth, or in seeing the most current Forth products available are invited to attend this day at no charge.

**The registration fee includes** all sessions, meals, and the Conference papers. Lodging is available at local motels or in the UR dormitories. Registration will be from 3–11 p.m. on Tuesday, June 11th, and from 8 a.m. Wednesday the 12th in Wilson Commons. There is an hourly shuttle to the airport during registration and checkout. Sessions will be held in Lower Strong Auditorium, and the open day will be in Wilson Commons.

**Those wishing to present papers** should submit an abstract by March 30th, and a final paper by April 30th. Papers longer than four single-spaced pages will be submitted to the **Journal of Forth Application and Research** for review, while shorter papers and abstracts will appear in their submitted form in the Conference Proceedings. Abstracts and papers should be sent to the Conference Chairman, Lawrence P. Forsley, Laboratory for Laser Energetics, 250 East River Rd., Rochester, New York 14623.

**For more information,** call or write to Maria Gress at the Institute, 70 Elmwood Ave., Rochester, NY 14611. Phone: 716-235-0168.

---

## Registration Form

Payment must accompany registration. Checks should be payable to the Rochester Forth Conference. Please mail your registration by May 15th to the Institute for Applied Forth Research, Inc., 70 Elmwood Avenue, Rochester, New York 14611.

Name ______________________________

Address ______________________________

______________________________

______________________________

Telephone ______________________

Registration: $325, Full-time students $200. ______

Dormitory Housing, 5 nights: Single $100, Double $75. ______

______ Prefer non-smoking roommate ______ Vegetarian meal option

______ I am unable to attend. Please send information on obtaining the Conference Proceedings.

Total Enclosed: ______________________

# Selected Publications

The Forth Interest Group Order Form (on the reverse side of this page) has two newly added publications selected by the *FIG* Publication Committee:

Forth Tools and Applications
1984 *FORML* Proceedings

**Forth Tools and Applications**

*by Feierbach and Thomas*

Here is a book on Forth written for the programmer who knows the language and now wants to learn how to use it. Provided in this book are the standard tools required to quickly create and debug Forth-base applications.

FORTH tools and Applications first gives you the advice you need to design your own programs. It also presents a number of techniques to aid you in the debugging process. Finally, you'll take a close look at a variety of utilities that can become a part of your final application — utilities such as file data selection, sorting and merging, binary search, and floating point.

Throughout this book the 79 Standard dialect is used with fig-FORTH words appearing where no 79 Standard is available. An appendix at the end of the book completely covers both dialects for your ease of reference.

As you master the tools of Forth programming, this book helps you develop good programming, documentation, and debugging skills.

Discover the fun of programming your own software systems with this popular new machine-free language. Let FORTH tools and Applications be your guide to the countless applications open to you with FORTH programming.

**1984 FORML Proceedings**

The proceedings from the 1984 Forth Modification Laboratory are now available and may be purchased from the Forth Interest Group. The FORML Conference, held each November at Asilomar, California, is an opportunity for serious Forth programmers to share new applications and ideas with the attendees and programmers around the world through the publication of the proceedings. Order today and make the 1984 FORML Proceedings a part of your Forth library.

**All Memberships Expire in April**

Don't forget to renew your FIG membership. If you haven't already renewed, return the renewal envelope (in this issue) to receive your next Forth Dimensions without a lapse in membership.

All members will receive membership cards which allow a 10% discount on publications sold by FIG. Your member number will be required to receive the discount, so keep it handy.

Foreign members should note the new pricing schedule on the order form. In addition to the foreign air rate, FIG is now offering a surface rate for shipments to countries outside the US, Canada and Mexico. Please allow extra delivery time for surface orders.

# FORTH INTEREST GROUP MAIL ORDER FORM

**P.O. Box 8231 San Jose, CA 95155 (408) 277-0668**

**Please Print**

**Name** ______________________

**Company** ______________________

**Address** ______________________

**City** ______________________

**State/Prov.** __________ **ZIP** __________

**Country** ______________________

**Phone** ______________________

**Membership in the FORTH Interest Group**
& Volume 7 of FORTH Dimensions
No sales tax, handling fee or discount on membership. See col.3. ☐

## BACK VOLUMES

| | | |
|---|---|---|
| Volume 1 FORTH Dimensions | *$15/16/18 | _____ |
| Volume 2 FORTH Dimensions | 15/16/18 | _____ |
| Volume 3 FORTH Dimensions | 15/16/18 | _____ |
| Volume 4 FORTH Dimensions | 15/16/18 | _____ |
| Volume 5 FORTH Dimensions | 15/16/18 | _____ |
| Volume 6 FORTH Dimensions | 15/16/18 | _____ |

## REFERENCE

| | | |
|---|---|---|
| FORTH 83 Standard | $15/16/18 | _____ |
| FORTH 79 Standard | 15/16/18 | _____ |
| Bibliography of Forth References, 2nd Ed. | 15/16/18 | _____ |

**OFFICE USE ONLY**

By ________ Date ________ Type ______
Shipped by ____________ Date ________
UPS Wt. ____________ Amt. ________
USPS Wt. ____________ Amt. ________
BO ________ Wt. ________ Amt. ______

## BOOKS ABOUT FORTH

| | | |
|---|---|---|
| All About FORTH | *$25/26/35 | _____ |
| Beginning FORTH | 17/18/21 | _____ |
| Complete FORTH | 16/17/20 | _____ |
| FORTH Encyclopedia | 25/26/35 | _____ |
| FORTH Fundamentals, V.1 | 16/17/20 | _____ |
| FORTH Fundamentals, V.2 | 13/14/16 | _____ |
| FORTH Tools | 19/21/23 | _____ |
| Mastering Forth | 18/19/22 | _____ |
| Starting FORTH (Soft Cover) | 20/21/22 | _____ |
| Starting FORTH (Hard Cover) | 24/25/29 | _____ |
| Thinking FORTH (Soft Cover) | 16/17/20 | _____ |
| Thinking FORTH (Hard Cover) | 23/25/28 | _____ |
| Threaded Interpretive Languages | 23/25/28 | _____ |
| Understanding FORTH | 3/4/5 | _____ |

## CONFERENCE PROCEEDINGS

| | | |
|---|---|---|
| FORML Proceedings 1980 | $25/28/35 | _____ |
| FORML Proceedings 1981 (2 V.) | 40/43/48 | _____ |
| FORML Proceedings 1982 | 25/28/35 | _____ |
| FORML Proceedings 1983 | 25/28/35 | _____ |
| Rochester Proceedings 1981 | 25/28/35 | _____ |
| Rochester Proceedings 1982 | 25/28/35 | _____ |
| Rochester Proceedings 1983 | 25/28/35 | _____ |
| Rochester Proceedings 1984 | 25/28/35 | _____ |

## JOURNAL OF FORTH APPLICATIONS AND RESEARCH

| | | |
|---|---|---|
| Journal of FORTH Research V.1 #1 | $15/16/18 | _____ |
| Journal of FORTH Research V.1 #2 | 15/16/18 | _____ |
| Journal of FORTH Research V.2 #1 | 15/16/18 | _____ |
| Journal of FORTH Research V.2 #2 | 15/16/18 | _____ |

## REPRINTS

| | | |
|---|---|---|
| Byte Reprints | $3.50/4/5 | _____ |
| Popular Computing 9/83 | 3.50/4/5 | _____ |
| Dr. Dobb's 9/82 | 3.50/4/5 | _____ |
| Dr. Dobb's 9/83 | 3.50/4/5 | _____ |
| Dr. Dobb's 9/84 | 3.50/4/5 | _____ |

## HISTORICAL DOCUMENTS

| | | |
|---|---|---|
| Kitt Peak Primer | $25/27/35 | _____ |
| fig-FORTH Installation Manual | 15/16/18 | _____ |

*PRICING

Column 1 - US, Canada, Mexico
Column 2 - Foreign Surface Mail
Column 3 - Foreign Air Mail

## ASSEMBLY LANGUAGE SOURCE LISTINGS

| | | |
|---|---|---|
| 1802 | *$15/16/18 | _____ |
| 6502 | 15/16/18 | _____ |
| 6800 | 15/16/18 | _____ |
| 6809 | 15/16/18 | _____ |
| 68000 | 15/16/18 | _____ |
| 8080 | 15/16/18 | _____ |
| 8086/88 | 15/16/18 | _____ |
| 9900 | 15/16/18 | _____ |
| ALPH MICRO | 15/16/18 | _____ |
| Apple II | 15/16/18 | _____ |
| ECLIPSE | 15/16/18 | _____ |
| IBM/PC | 15/16/18 | _____ |
| NOVA | 15/16/18 | _____ |
| PACE | 15/16/18 | _____ |
| PDP-11 | 15/16/18 | _____ |
| VAX | 15/16/18 | _____ |
| Z80 | 15/16/18 | _____ |

## MISCELLANEOUS

| | | |
|---|---|---|
| T-Shirt Size: _____ | $10/11/12 | _____ |
| Poster (BYTE Cover) | 3/4/5 | _____ |
| Handy Reference Card | FREE | ☐ |

| | |
|---|---|
| Subtotal | $________ |
| 10% Member Discount<br>Member No. Required ________ | ________ |
| Subtotal | $________ |
| CA residents add sales tax | ________ |
| Handling Fee | $2.00 |
| Membership (*$20/27/33) | ________ |
| Total | $________ |

VISA ☐ M/C ☐ # ____________________
Expiration Date ________

$15 minimum on all VISA/MC orders.

Make check or money order payable in US funds drawn on a US bank to: FIG.

PAYMENT MUST ACCOMPANY ALL ORDERS (Including Purchase Orders)
ALL Prices include Shipping.

*Pollard's*

# Monte Carlo Factorizer

*Nathaniel Grossman*
*Los Angeles, California*

The factoring of integers is an old mathematical problem: given an integer $N \geq 2$, the goal is to find two other integers R and S, each $\geq 2$, such that $N = RS$. If R and S exist, N is called *composite* and the integers R and S are (non-trivial) *factors* of N. If no such R and S exist, the integer N is called *prime*. The so-called Fundamental Theorem of Arithmetic states that every integer $\geq 2$ can be expressed as the product of prime integers and, except for the order in which the factors are listed, in one way only. There is considerable importance attached to the task of determining which integers are prime and of finding ways to express the remaining integers integers as a product of prime factors.

About the status of this problem there is good news and bad news. The good news is that there is an absolutely sure algorithm for testing primality and for factoring. (It turns out that there are complicated algorithms which test for primality relatively quickly if a factorization is not required for those numbers found to be composite.) If the number N has non-trivial factors, at least one of them must be no larger than $\sqrt{N}$. To find such a factor, it suffices to divide N by each integer greater than one and no larger than $\sqrt{N}$: if one of those trial divisions leaves no remainder, the divisor is a factor of N. If no trial division is without remainder, N is prime. The bad news is that this algorithm, sure to take no more than $\sqrt{N}$ divisions, may for certain N require essentially $\sqrt{N}$ divisions. For many integers of current interest, that many divisions is beyond the capability of today's fastest computers to complete within an acceptable time. Nevertheless, it is possible to attack large numbers with the hope of successful factoring by employing one of several strategies. This paper will show how one of those strategies can be implemented.

Those interested in an overview of the most intense efforts to factor very large numbers will find an excellent account of the principal methods and of the notable successes and failures in a recent article written for non-specialists by H. C. Williams.[5] My account of "wheels" and of Pollard's method is drawn from that article and from Pollard's original announcement.[4]

### Wheels

Before going to Pollard's method it will be worthwhile to describe a technique for improving upon the "naive" division test mentioned above. The difficulty there arose because an unacceptable number of divisions might be required to test an integer N and there was no way to predict in general whether testing of a "random" N would be easy, hard, or essentially impossible. If a scheme could be devised for substantially cutting down the number of trial divisions, the method might become acceptable, depending on the computer and the time available for carrying out the computation.

If space is available, the list of all prime numbers no larger than a certain integer L can be stored in memory. In performing the trial divisions on N, only the stored primes need be used as trial divisors among the integers $\leq L$. As the number of primes less than L is about L/log L, this can lead to a substantial saving in computation: only about 12% of the integers less than 10,000 and about 7.8% of those less than 1,000,000 are primes, compared with the 50% that are odd numbers.

If there is but a little storage available, savings can be made by means of an artifice that specialists call a *wheel*. As an example, note that the subject integer N can be tested easily for divisibility by 2, 3, and 5, and such factors removed. It is easy to see as a consequence that a trial divisor D need not be considered unless the remainder when D is divided by 30 can be found in the list 1, 7, 11, 13, 17, 19, 23, 29. (All the remaining remainders are divisible by at least one of 2, 3, or 5.) This restriction reduces the number of trial divisions by a factor of 8/30 to about a quarter of the original. The eight numbers 1, ... 29 are the "spokes;" after the spokes are run through, the wheel is given another "turn" by addition of the product 30. Wheels with a great number of spokes have been used successfully. Note that the proportion of savings from use of a wheel is constant no matter how long the sequence of trial divisors is to run.

### Pollard's Method

Trying all suspect divisors or just a proportion of them is not practical if the integer N is even "moderately" large. Pollard proposed two methods for selecting more promising divisors and thereby cutting back on the number of trials. These methods are especially useful nowadays when computers take the drudgery (if not the waiting) out of numerical computations. Trials at factoring that end in failure can be put up with because after the introduction of the new, ingenious factorization methods, "*several failures in a row were of no particular importance,* as long as they were followed by at least one success." (Morrison and Brillhart[3]) One of Pollard's methods, called first by him the *ρ-method* and then the *Monte Carlo method*, is well suited to implementation on a personal microcomputer. I have implemented a (slightly simplified) form of the Monte Carlo method on an Apple IIe computer in the Forth language.

The idea behind the Monte Carlo method is quite simple.[4,5] Suppose the integer N to possess a prime factor p that we seek. Fix a non-linear polynomial f(x) with integer coefficients. Choosing a starting integer $x_1$, generate $x_2, x_3, \ldots$ by setting $x_k = f(x_{k-1}) \bmod N$. The x-values may be well scattered throughout the interval from 0 to N-1. If, however, the x's are again reduced mod p and if p is "small," the newly reduced x's will begin to pile up in the interval for 0 to p-1. If f(x) is also chosen to have "random" output, it is to be expected that soon two of the x's will be equal mod p. Of course p is unknown, but the two x's, say $x_i$ and $x_j$, will be such that $x_i - x_j$ is divisible by p. Thus p will be a divisor both of $x_i - x_j$ and N, so that p will be found as a factor of the greatest common divisor

$\gcd(x_i - x_j, N)$.

The words "random" and "Monte Carlo" suggest that some probabilistic engine is at work. That is not so: what is required is that f(x) spread the values generated widely and "uniformly" throughout the integers less than N. Whatever that means, it turns out that any quadratic polynomial $f(x) = x^2 + b$ seems to work satisfactorily provided that $b \neq 0, -2$. (Linear functions are not suitable.)

If the newly-generated x's were to be compared with each of the earlier-generated x's, the resulting storage and retrieval and the necessary computations of greatest common divisors would complicate and prolong the calculations to no clear benefit. Instead, the algorithm is pared down and simplified - all that matters is that there be at least one success! Start with $x_1$ and generate $x_2 = f(x_1)$. Then begin cycles composed as follows: supposing the pair $(x_i, x_{2i})$ available at the beginning of the $i^{th}$ cycle, complete the cycle by computing $(x_{i+1}, x_{2i+2})$ and $\gcd(x_{2i+2} - x_{2i}, N)$. Because $x_{i+1} = f(x_i)$ and $x_{2i+2} = f(f(x_{2i}))$, the production of the new pair is straightforward, especially because the calculations are carried out mod N. The cycles roll on until the greatest common divisor takes a value $R > 1$. Then the calculations will stop because R is a factor of N greater than one. Unfortunately, it can happen that $R = N$.

Pollard[4] gives estimates for the number of cycles required to factor N. If p is the smallest divisor of N, Pollard builds on results reported by Knuth to show that the expected number of cycles needed to produce a non-trivial factor of N is about $1.0308\sqrt{p}$.

As an example, with the polynomial $f(x) = x^2 + 1$ and $x_1 = 6$ and with the implementation described below, the decomposition into two prime factors

$$4{,}225{,}910{,}033 = 65{,}003 \times 65{,}011$$

appears after 335 cycles. (The Pollard expected number of cycles is 263.)

In Pollard's formulation, the Monte Carlo method is slightly more complicated than I have described it here. (I give the full description in my discussion below of screen #4.) Using a further small extension of the method and a lot of mulit-precision arithmetic, Brent and Pollard[1] factored the eighth Fermat number, $F_8 = 2^{256} + 1$, into the product of two prime factors. Although $F_8$ has seventy-eight decimal digits and the smaller of the two prime factors has sixteen, the calculation was performed on a Univac 1100/42 computer in just two hours of time.

(The Fermat numbers $F_n = 2^{2^n} + 1$ are primes for $n = 0,1,2,3,4$. Fermat conjectured the primality of all $F_n$, but Euler showed the divisibility of $F_5$ by 641. Unfortunately, $F_5$ is just past the range of the present implementation, which factors numbers $N < 2^{32}$.)

**Implementation of Pollard's Method**

The Monte Carlo method rests upon one hypothesis: that the number N to be factored has a "small" prime factor. Suppose, for the sake of argument, that $N < 2^{32}$. Then, if N factors, it must be divisible by a prime $p < 2^{16}$, and Pollard's estimate predicts that the algorithm should be expected to find a factor of N divisible by p in no more than $2^8$ cycles. To run the algorithm on a personal computer such as an Apple II under such circumstances will entail no intolerable wait for the results. Suppose, however, that $2^{16} \leq N < 2^{32}$, and that N is a prime. If N is close to $2^{32}$, we might expect to wait about $2^{16}$ cycles for the results. However, there is no need to wait so long. If N were not a prime, it would be factorable and factors would be discovered in far fewer steps. (Pollard estimated from numerical experiments that the number of cycles needed to find the smallest prime factor p would be $> 2\sqrt{p}$ with probability about 0.065.)

If the number N to be factored is "large," there is no *a priori* reason to expect a "small" prime factor. For that reason, Williams[5] calls Brent and Pollard "lucky" in their attempt to factor $F_8$. For the same reason, there seems no pressing need to implement the Monte Carlo method to higher precision on a personal computer. Those whose profession it is to factor large numbers—number theorists, cryptographers, and specialists in the design of computations—will have access to the giant number-crunchers. It seems reasonable, therefore, to set the goal of factoring numbers $<2^{32}$. This modest goal can be reached with the aid of the ordinary Forth language tools enriched by the Double-Number Extension Word Set, provided that two or three further words are added to the dictionary so that certain double-quadruple operations can be carried out. Those who carry out the implementation will get an inside look at this interesting factorizer, an algorithm that, within its clear limitations, works well.

My implementation is given in the screens #0 through #8. The documentation in screen #0 extends into screen #1 and is not part of the program. In fact, there is something missing: definitions for the words **DUM***, **DUM/**, and **DUM/MOD**, the actions of which are specified on screen #0. I have given there suggestions on how to supply these words. Those who have floating point extensions to their Forth Systems may find equivalents contained therein. (I use the version furnished with the floating-point extensions to MicroMotion MasterFORTH. They are written in 6502 assembler and, consequently, they are fast.)

Screen #2. The algorithm is built on the quadratic interative polynomial $f(x) = x^2 + b$. The numbers x and b are always to be double integers. To give leeway for experimentation, b is stored in the 2-variable **INCREASE_SQUARE** with the default value 1. This gives the default function: $f(x) = x^2 + 1$. **SQ(MOD)*** computes f(x) mod N and **NEW_PAIR** turns the pair $(x_i,x_{2i})$ into the pair $(x_{i+1},x_{2i+2})$.

Screen #3. There is some interest in keeping count of the number of cycles; that number is stored in the variable **COUNT**. The variable **AT_MOST** furnishes the only non-destructive way to halt the onrushing cycles. It is set to the default value 1000.

Screen #4. The word **DGCD** computes the greatest common divisor of two double integers held on the stack. It is based on the word of the same name in my paper[2] "Long Divisors and Short Fractions," but with **DUM/MOD** replacing the word **UD/MOD** defined in that paper.

The greater share of the calls to **DUM/MOD** occur within **DGCD**, so that the running time of the algorithm might sometimes be reduced by Pollard's artifice of storing products

$$\Pi_m = \prod_{i=1}^{m} (x_{2i} - x_i) \bmod N$$

and testing $\gcd(\Pi_m,N)$ only for selected m—he suggests those m divisible by 100. In the present case, the algorithm may run only two or three cycles before announcing factors of N. For large numbers such as $F_8$, it may be assumed that preliminary trials have searched for and failed to find "small" prime factors. In such cases, introduction of $\Pi_m$ and resulting manipulations makes sense, and will lead to substantial savings in computing time. Here, programming it does not seem worth the bother.

Screens #7 and #8. Here are two versions of the Monte Carlo algorithm differing only in their display. Each expects the double integer N on the stack and each returns a message either

```
SCR # 0
\ Pollard's Monte Carlo Factorizer          FORTH-83  19AUG84NG

   This file contains a method for factoring integers published
by J. M. Pollard in BIT 15(1975), 331-334.
   The implementation calls on several extensions of the Double-
Number Extension Word Set from the Forth-83 Standard:
 DUM* expects two unsigned double factors on the stack and
pushes their unsigned quadruple product onto the stack ---
 DUM/ expects a uq dividend inside a ud divisor and pushes a ud
quotient onto the stack.  (User must assure suitable data to
give a genuine ud quotient.) ---
 DUM/MOD is similar to DUM/, leaving a ud rem inside a ud quot.
   Such words might be built from D. A. Beers' "Quadruple Word
Simple Arithmetic" in Forth Dimensions IV/1, p. 17.  The word
UD/MOD given by N. Grossman, "Long Words and Short Fractions,"
Forth Dimensions (to appear), could be used to [cont on scr #1]
SCR # 1
\ loader                                    FORTH-83  19AUG84NG

: MARKER   ( null action)   ;      2 9 THRU

\ [cont. from scr #1]
\ write DUM/ and DUM/MOD.  Users of MicroMotion MasterFORTH
\ will find DUM*, DUM/, and DUM/MOD in the floating point exten-
\ sion, located in the file DOUBLE and written in 6502 assemb-
\ ler.  They should add the files ASSEMBLER and DOUBLE to the
\ dictionary.
\    This screen loads the file POLLARD and makes available the
\ factoring words FAC and FAC'.  The sequences  <ud> FAC or
\ <ud> FAC'  result in printing either (not necessarily prime)
\ factors of <ud> or a warning that the cycles have reached in
\ number the upper limit stored in the variable AT_MOST (scr #3)
\ without finding factors.  The limit is arbitrarily made 1000.
```

stating factors of N or reporting that the algorithm has **QUIT** after carrying out **AT_MOST** cycles. If N is a prime and factors are reported, they will be N and 1. In between the startup and the message, the display will report progress: **MONTE_CARLO_FACTOR'** shows the triples i, $x_i$, $x_{2i}$ scrolling upward and **MONTE_CARLO_FACTOR** displays anxiety-quelling dots marshalled into rows, one dot appearing for each cycle begun.

I find it convenient to enlarge the dictionary by aliases **FAC'** and **FAC** for those long names.

```
SCR # 2
 \ Sequence and pair generators                  FORTH-83  19AUG84NG

 2VARIABLE INCREASE_SQUARE    1. INCREASE_SQUARE 2!

 : SQ(MOD)+    \ d d1 --- d d2
    2DUP DUM*                       \ d1*d1
    5 PICK  5 PICK
    DUM/MOD 2DROP                   \ mod d
    INCREASE_SQUARE 2@  D+   ;  \ add offset

 : NEW_PAIR    \ d d2 d1 --- d d4 d3
    5 PICK  5 PICK  2SWAP
    SQ(MOD)+
    2SWAP  5 ROLL  5 ROLL
    SQ(MOD)+  SQ(MOD)+
    2SWAP 2DROP 2SWAP   ;
SCR # 3
 \ Conveniences                                  FORTH-83  19AUG84NG

  VARIABLE COUNTER   VARIABLE AT_MOST   1000 AT_MOST !
 2VARIABLE FIRST_X    6. FIRST_X  2!

 : SETUP_POLLARD    \ d --- d d2 d1
    0 COUNTER !  FIRST_X 2@  SQ(MOD)+  FIRST_X 2@   ;

 : INCREASE_COUNTER    \ ---
    1 COUNTER +!   ;

 : .COUNTER    \ ---
    COUNTER @  0 5 D.R   ;

 : .NEW_PAIR    \ d d2 d1 --- d d2 d1
    2DUP  12 D.R  2SWAP   2DUP  12 D.R  2SWAP   ;
```

Examples:

(i) 2,771 factors into 17 times 163 in 5 cycles.

(ii) 9,221 needs 235 cycles before it proclaims its factorization into 9,221 times 1 (it is prime).

(iii) 314,159,265 is factored into 315 times 997,331 (neither prime) in two cycles.

(iv) The Mersenne number $M_{29} = 2^{29} - 1 = 536,870,911$ is broken down into 486,737 times 1103 in 13 cycles.

(v) 1,234,567,891. **FAC** returns 1,000 dots followed by the message "After 1000 cycles, no factor of 1234567891 has been found."

(vi) 4,225,910,033. **FAC** returns 335 dots and the message "The number 4225910033 factors into 65011 times 65003 (335 cycles)" and runs in about 40 seconds.

(vii) If N is prime, **FAC** will eventually return N and 1 as its factors provided no more than **AT_MOST** cycles run. However, returning N and 1 does not mean that N must be prime. 1,007. **FAC** returns the factors 1007 and 1 after 5 cycles, but

**13.** **INCREASE_SQUARE 2!** changes the iteration function to $x^2 + 13$, after which 1007. **FAC** returns factors 53 and 19 in 2 cycles. Also, use of $f(x) = x^2 + 17$ returns factors 19 and 53 in 3 cycles. ("*several failures in a row were of no particular importance,* as long as they were followed by at least one success.")

## REFERENCES


1. Brent, R.P., and J.M. Pollard, Factorization of the eighth Fermat number, *Mathematics of Computation 36* (1981),627-630.
2. Grossman, N., Long divisors and short fractions, *Forth Dimensions* VI/3(1984), 10-19.
3. Morrison, M.A., and J. Brillhart, A method of factoring and the factorization of $F_7$, *Mathematics of Computation 29* (1975), 183-205.
4. Pollard, J.M., A Monte Carlo method for factorization, *BIT 15* (1975), 331-334.
5. Williams, H.C. Factoring on a computer, *The Mathematical Intelligencer 6* (1984), no. 3, 29-36.

```
SCR # 4
\ Greatest common divisor                    FORTH-83  19AUG84NG

: DGCD   \ d1 d2 --- d ( = double number gcd of d1 and d2 )
  BEGIN               \ the euclidean algorithm
   2SWAP 2OVER
   >R >R 0. R> R>
   DUM/MOD  2DROP
   2DUP D0=
  UNTIL   2DROP   ;

: DGCD_STACKER    \ d d2 d1 --- d d2 d1 d3
\ pushes gcd of d2-d1 and d onto stack as a double
  2OVER 2OVER  D- DABS
  7 PICK  7 PICK
  DGCD   ;
```

```
SCR # 5
\ Conveniences                               FORTH-83  19AUG84NG

: IS_GCD_>_1?    \ d d2 d1 d3 --- d d2 d1 d3 f
   2DUP  1.  2SWAP  DU<   ;

: DU.    \ ud. ---    ; print an unsigned double number
   <# #S #>    TYPE SPACE   ;

: .FACTORS    \ d d2 d1 d3 ---
   2SWAP 2DROP 2SWAP 2DROP
   2OVER CR
   ." The number  " DU. ."   factors into   " 2DUP DU.
   ."  times  " 0. 2SWAP DUM/ DU.   ;
```

```
SCR # 6
\ Warning                                    FORTH-83  19AUG84NG

: ROADRUNNER
  BEEP BEEP   ;

: .WARNING    \ d d3 d2 d1 ---
  CR CR
  ." After " AT_MOST @ .
  ." cycles, no factor of  "
  2DROP 2DROP 2DROP DU.
  ."  has been found.  "
   ROADRUNNER   ;
```

```
SCR # 7
\ Pollard's Monte Carlo Factorizer'          FORTH-83  19AUG84NG

: MONTE_CARLO_FACTOR'     \ d ---
   SETUP_POLLARD  0.           \ initialize counter and stack
   BEGIN
    INCREASE_COUNTER
    AT_MOST @  COUNTER @ U<    \ too many cycles?
    IF   .WARNING  QUIT   THEN
    2DROP
    CR .COUNTER
    NEW_PAIR  .NEW_PAIR
    DGCD_STACKER  IS_GCD_>_1?
   UNTIL
   ROADRUNNER
   CR CR  .FACTORS  CR CR   ;

SCR # 8
\ Pollard's Monte Carlo Factorizer           FORTH-83  19AUG84NG

: MONTE_CARLO_FACTOR    \ d ---
   SETUP_POLLARD  0.  CR   \ initialize counter, stack, display
   BEGIN
    INCREASE_COUNTER
    AT_MOST @  COUNTER @  U<    \ too many cycles?
    IF  .WARNING  QUIT  THEN
    2DROP  ASCII . EMIT        \ print . for watcher's sake
    NEW_PAIR
    DGCD_STACKER  IS_GCD_>_1?
   UNTIL
   ROADRUNNER
   CR CR  .FACTORS  CR CR
   ASCII ( EMIT  32 EMIT  COUNTER @ .  ." cycles )"   ;

SCR # 9
\ Short aliases

: FAC'   MONTE_CARLO_FACTOR'   ;

: FAC    MONTE_CARLO_FACTOR    ;
```

**TDS900 C-MOS FORTH COMPUTER** **$230**

Single board Eurocard FORTH computer. Entirely C-MOS, consumes typically 16mA. Uses 6303 8/16-bit processor with hardware multiply, timer, and serial interface. Unused op-code trap forces processor back to FORTH. Has 12K RAM and an additional 8K spare socket. Most memory can use EPROM in place of RAM for dedicated applications. Fig-FORTH language with many other useful real-time words added and a full-screen editor working with dumb CRT terminals. Data, address and control buses are brought to the 64-pin indirect connector. The system is expandable by use of 6800, 6300 or 6500 series peripheral devices.

**TDS901-X NON-VOLATILE FORTH COMPUTER** **$294 to $549**

A TDS900 incorporating non-volatile RAM and modified FORTH system. FORTH applications and/or stored data are retained in NV-RAM when power is removed. Invaluable while developing software. Substitute the following in place of -X to get the order code:- -2K, -4K, -6K, -8K, -10K. This refers to the amount of non-volatile memory. The rest, up to 12K, is normal C-MOS RAM.

**TDS902 64K C-MOS FORTH COMPUTER** **$536**

Although the 64K memory space is almost full the card takes only 25mA typical because of its all C-MOS design.

**TDS900-ASS 6303 FORTH ASSEMBLER** **$115**

Software for any of the TDS900 series computers which allows FORTH words to be defined in microprocessor code. Includes structured words such BEGIN/UNTIL compiling to machine code.

**TDS950 COMBINATION CARD** **$189**

Cassette interface card containing 1200 baud audio to RS232C converter, line transformer/power supply (115/230V) for itself and a TDS900 computer, and 200mA-Hr of back-up battery to make a TDS900/TDS950 combination self-contained and portable. This is an ideal starting system. FORTH software for cassette load/dump.

**TDS971 COMMUNICATIONS CARD** **$230**

Contains an IEEE-488 Talker/Listener interface and two RS232C.

**TDS972 PARALLEL/SERIAL/A-D CARD** **$189**

8-bit parallel I/O, RS232C and 8-bit A-D converter.

**This is an extract from our brochure of industrial FORTH cards. Brochure free. Official orders accepted. For private purchases send cheque. We accept most currencies. Dly 2-3 weeks.**

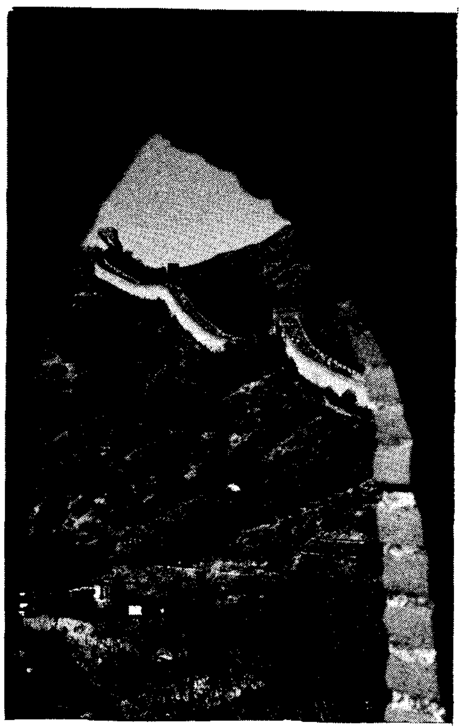
**Triangle Digital Services Ltd.**
100a Wood Street, Walthamstow, London E17 3HX
Tel: 01-520-0442 Telex 262284 (Quote REF 775)

*Part Two*

# FORML China Tour 1984

**10-3:** We arose early enough to place our leaden bags in the hallway by 6:30 a.m., just before departing from Hong Kong. We met our first guide from the People's Republic of China, a young woman working for CITS, China's official travel service. She swept us to Tai Kok Tsui pier, where we waited in line to board the hover-ferry to Guangzhou (formerly Canton). Travel inside China is handled by CITS according to availability of rooms and flights. Because no flights were available out of Hong Kong on this day, we were able to travel a scenic route along waterways which were legendary long before Clavell popularized them in his *Taipan*. And although Guangzhou was not on our itinerary, after a cursory customs inspection, local guides bussed us through a brief tour of the city. At the large center where international trade fairs are held twice annually, we ate the first of many meals which would move us to gustatory extremes. We then changed U.S. travellers checks and Hong Kong dollars into Chinese yuan and succumbed to another buyers' frenzy before catching our flight to Shanghai.

We lumbered off the plane (only one checked bag per person meant many hand-carried kilos; and we had been fed again, in flight) to breathe the first cool air of the trip. Night was advancing, but the city cast only a dim glimmer to penetrate the leafy boulevards. Night stollers in Shanghai, we were told, benefit from an ordinance which forbids lighting brighter than the full moon. We were already far from glitzy Hong Kong! During our bus ride to the Shanghai Hotel, we found that the local CITS guides and our hosts at Shanghai's Jiao Tong University had both planned for our arrival. Late plans were formulated over drinks before we fell, exhausted, into bed.



— *George W. Shaw II*

**The Great Wall crowns the highest passes and steepest inclines.**

**10-4:** Our first full day in the People's Republic of China was spent getting to know the people and places of Shanghai. Mr. Hu, our competent local guide, escorted us everywhere and explained everything to us. We stopped by a public park near the river, where some elderly Chinese, engaged in a dual form of t'ai chi, attracted us almost as quickly as our appearance drew a crowd of polite and enthusiastic citizens. We had the pleasure of speaking in English with a few students and professionals, but smiles and appreciative gestures were as plentiful as the people themselves.

Afterward, we visited famous Yuyuan Gardens, a two-hectare walled estate in the placid architectural style of the wealthy Chinese of past centuries. It is now open to the public, and has proved so popular that the tremendous congestion there came to exemplify for us the twelve million inhabitants of Shanghai. Outside the five dragon walls, we saw won-ton being made in shops festooned with bright silk banners commemorating the anniversary of the revolution. We stopped at a factory, where workers sit side by side before rows of tapestry looms, cutting and clipping knots by hand at a fierce pace. We also visited a jade plant, where workers sit at individual work stations with power drills to carve goddesses and goblets from lucent chunks of stone; and where, upstairs, the raw material is ivory. We were very impressed by the quantity and consistent high quality of the finished pieces. Then, just to make sure we had an eventful day, our local guide arranged an evening at the circus for us!

— *George W. Shaw II*

**One of the dragon walls enclosing Yuyuan Gardens in Shanghai.**

**10-5:** This morning, part of the FORML group went to Jiao Tong University to be met by the university's President and an audience of about two hundred. During this precursor to the FORML conference, which was to begin in a few days, Forth was introduced to those who hadn't used it yet, and a few advanced concepts were explored.

While those ceremonies were taking place, the rest of the group was taken to a silk-printing factory to learn a bit about the process. After a brief foray into the company store, all re-boarded the bus for a visit with some people who will be important to the PRC's future: its kindergartners.

— George W. Shaw II

**A group presentation at Shanghai's Jiao Tong University.**

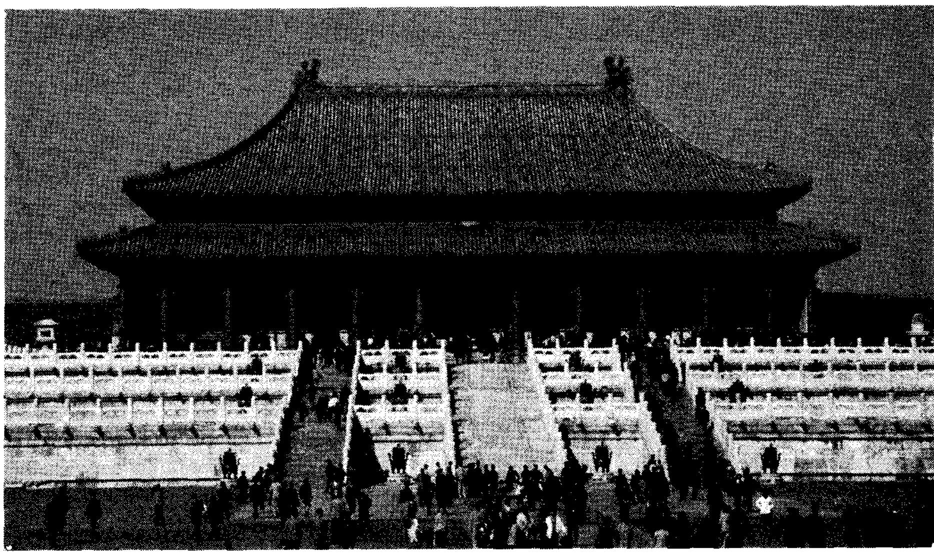
The playground was full of four- to six-year-olds, and when the bus pulled to a stop they rushed the fences, repeatedly calling, "Hi! Hi!" The moment of pandemonium was brief, as the six-year-olds filed out to greet each of the American "aunts and uncles." By the time each of them had adopted one adult with the clasp of a proprietary hand, the younger children had disappeared. We found ourselves being escorted to a room in the school where each of us was served make-believe tea with styrofoam cakes — these youngsters had obviously been learning how Western children play. After tea-time, we were again led by the hand, this time to a room where the four-year-olds were making origami gifts for us — paper boats, hats and flapping birds which have taken their place in photo albums alongside snapshots of that day. The final event was a musical show in which most of the children sang, played instruments (we heard "Chopsticks" played as a piano duet) and danced; and they performed a colorful beach-ball-and-panda-suits choreography that conclusively stole our hearts as well as the show. Our young, panda-clad hosts then took us by the hand once more and led the way to the waiting bus. As we pulled away, young voices called out, "Bye! Bye!"

We were let out at a housing complex next. We visited the apartment where a sixty-one-year-old woman lives with her husband, daughter, son-in-law and grandchild. This family lives relatively comfortably in a two-room apartment with a shared, walk-down bath. We were offered tea and cigarettes (modern gestures of hospitality), and passed a pleasant time chatting about lifestyle and income (this household earns about 300 yuan per month, of which fifteen go for rent, 140 for food; most homes have a black-and-white television — a color set costs a year's gross wages).

Lunch was served back at the Shanghai Hotel, but four members of the group abandoned hearth, hotel and tour guides to find vegetarian cuisine. Meatless cooking was, somewhat surprisingly, rare throughout this trip. The best was found in the Jade Buddha Temple, a working temple and place of pilgrimmage, where the monks prepare wonderfully varied and tasty meals from vegetables, mosses, mushrooms and other fungi. In other parts of the temple, the respectful remove their shoes before entering rooms where larger-than-lifesize jade statues representing the Buddha are seated in repose on the principles of universal law.

Later that evening, the President of Jiao Tong University of Shanghai gave a banquet in honor of the FORML tour group. It was an occasion of ceremony and friendship, and we shared in a delightful repast which few in the United States are able to sample, including eels, young sparrows and sea cucumbers.

**10–6:** "Above there is heaven, below there are Hangzhou and Suzhou." Boxed breakfasts and a four-hour train ride brought us to lovely Hangzhou.



— George W. Shaw II

**One of the major palaces in the Forbidden City.**

The staff of the Hua Jia Shan Guest House was smilingly helpful, and worked very hard at communicating with their guests in English and Japanese. On the shores of Xi Hu (West Lake), the population of one million seems in harmony with its natural surroundings, in contrast to Shanghai's crowds of commuting bicyclists. The hotel grounds are laced with ponds, winding pathways and willow trees.

**10–7:** We spent a relaxing day touring the West Lake area. While much of Hangzhou was destroyed during last century's Taiping Rebellion, and although modern China seems slightly disdainful of its spiritual heritage, many legendary sights have been restored to a fine degree. Lingyin Temple is the only working Buddhist monastery in this region, and is visited by so many devotees that a small but thriving trade in souvenirs and religious articles has sprung up nearby. Adjacent to the temple, the "hill that flew from afar" is still covered with hundreds of stone carvings that were intended to help anchor the hill in place, it having already flattened one village. On a bluff in the near distance, the Six Harmonies Pagoda was erected in 970 A.D. so that its geomantic force would alleviate the river's devastating tidal bore.

An afternoon cruise on West Lake and a visit to a fan factory capped the day. Tour participants scheduled to lecture at the Shanghai conference were packed off on another train, while those who remained settled for a very welcome early retirement.

**10–8:** Tour members who remained at Hangzhou embarked this morning on a train to Wuxi, where they were lodged in a small guest house of their own, with luxurious terraces overlooking the agricultural lake country which surrounded it.

Meanwhile, the Forth experts in Shanghai were greeted by Professor Zhou, who introduced each U.S. delegate, described FIG's purpose and summarized Forth's history worldwide and at Jiao Tong University, where it has been used since 1980 under the gracious auspices of Dr. Zhu, Director of the Computer Center. FIG President Robert Reiling responded in kind, thanking the hosts for their interest in Forth and in becoming a FIG Chapter.

Bill Ragsdale, founding President of FIG, used human language as an analogy for understanding computer languages, and described how both can reflect our thought processes. He explained that large problems can be stated very simply, that a problem's solution is usually arrived at by breaking it into manageable components, and that the environment must accomodate the solution. He emphasized that FIG derives much of its growth from volunteer efforts, as compared to vendor-driven or manufacturer-driven organizations. This may result from the way in which Forth meets the real needs of individual users.

Charles Moore continued the program by elaborating on Forth's growth without corporate or governmental backing. He described Forth as a creation of the programmers, who now are faced with the prospect of convincing management to allow them to work with the tools of their own choice. He views Forth as a versatile programming language with advantages of the newer special-purpose languages (e.g., Smalltalk, Modula-2, Occam, Prolog) implicit to it, but without their limitations. Mr. Moore then enumerated some major applications of Forth in fields like radio telescopy, optical telescopy, gas chromatography, business systems, commercial telephone systems, special-effects photography, robot control and health care (administrative and technical uses), among many others.

Later that afternoon after discussion of F83, the Laxen/Perry implementation of Forth-83, Kim Harris spoke on problem-solving with Forth. Using a sample problem of programming a camera for proper exposure, he discussed the classic Forth technique of top-down problem refinement and bottom-up coding. The first solution is not always the best solution, so familiarization, expansion and reduction/factoring were introduced to the audience. This talk concluded with a review of the elements of Forth that make it so suitable to this methodology, and with a sample high-level solution to the posited problem.

**10-9:** Bill Ragsdale opened the day with his discussion of the Forth virtual stack machine, of the differences between Forth standards, and with a demonstration of Forth on the Radio Shack Model 100 (the only computer to accompany the group). In response to a question from the previous day, George Shaw then detailed differences between Forth and COBOL, followed by Robert McGhee with a comparison of twelve computer languages evaluated according to fifteen criteria.

*— George W. Shaw II*

**"Store ahead"?**

Following several other talks, many of which are documented in other FORML publications (e.g., in the upcoming *1984 FORML Proceedings*), all that remained was an exchange of mutual thanks and appreciation for the hosts and attendees of the conference. FIG representatives sincerely offered to help Forth grow in the People's Republic of China, and hope to have many more opportunities to do so.

**10-10:** After a late reunion on the previous evening, the entire group embarked via plane for Beijing, capitol of the vast nation. After a short delay at the airport, the group found its accomodations at the ultra-modern Lido Hotel, where our local guide appeared. In no time, all were whisked off to People's Square, with its massive monument to Mao Zedong. The 15th century Temple of Heaven was the next stop, site of imperial Chinese divination and harvest-related prayer. There, the natural cycles of seasons and stars were woven by the Son of Heaven into a tapestry of great benefit to all the people. Today, the temple is a museum and tourist attraction, reminding a few, perhaps, of times when the laws of nature were of as widely recognized import to daily life as the laws of society.

**10-11:** FORML members awoke to another opportunity to address a distinguished audience, this time at Beijing's Tsinghua University. About 220 people attended this meeting, half of whom had seen Forth in operation. The Forth virtual stack machine was discussed, as was the history of FIG and FORML. Charles Moore drew amused laughter with his comment, "Forth is a revolutionary language; maybe it is appropriate that it be popular in a revolutionary country such as China." After the formal event concluded, the tour group was treated by the local guides to an evening at the Chinese opera.

**10-12:** Early morning found a few of us worn out and feeling a touch of the respiratory infection that passes for Beijing's version of "turista." But one and all boarded a bus headed out of town for the Great Wall. Autumn had nipped the trees with russet colors, but sunlight warmed the breeze which blew over the ancient stones. Climbing the steep steps on the safe, sheltering side, and listening to the sound of the wind as it rushed in from vast distances — somewhere out there, you can still feel the presence of Tibet and Mongolia — it was not difficult to imagine a glow of campfires circling the world. As at Lingyin Temple, a small local industry has grown to serve the tourists, but here they mostly hawk sweatshirts, t-shirts and other memorabilia. The real commodity, though, is the presence of antiquity. These dusty peddlers aren't victims of a new economy: they have always been here. Before the revolution and t-shirts, they sold water and food to travellers, meager feed for their stock, shelter for the night; before that, they traded small pleasures and fortunes to the soldiers who garrisoned the Wall; and before that, they were conscripted to hew stone blocks out of the native hills. The Wall has been a way of life for a long time in China.

**10–13:** Our last day in Beijing saw us admiring very old astronomical instruments — appropriate activity for proponents of the computer language which received its baptism in some of the major observatories of the world. We also toured the Forbidden City, residence of past emporers. Many of the 9,000 rooms, which cover a moat-enclosed 250–acres, have been restored to their simple lines and elegant ornamentation. The day concluded with a small meeting, during which mutual interests and business opportunities for U.S. and Chinese organizations were discussed.

**10–14:** FORML tour members boarded a flight to Xi'an, capitol of Shaanxi Province and site of the now-famous terracotta army. There we were once again greeted by Professor Zhou, who eased our passage throughout the country, along with officials from Jiao Tong University of Xi'an and the Chief Engineer for the province.

The local guide took us to the Forest of Stone Tablets, a library devoted to the works of Confucius, the entirety of which was carved in stone along with commentary and annotations. After a brief tour of the city (population 2.5 million), which used to be the largest in the world, we witnessed a late-evening performance of Tang Dynasty dancing.

**10–15:** The day began with a much-anticipated trip to see restored portions of the terracotta army, buried in the third century B.C. around the tomb of the late emporer. The huge archeological site met all expectations, and the related museum was very helpful in explaining the history of the region. During this time, the group also viewed a partially excavated indigenous village and noted congruities with sites in the Americas.

Returning to Xi'an, our guide took us to the university and surrendered us into the hands of officials there. Like its counterpart in Shanghai, Jiao Tong University here is technically oriented; it has six research institutions, fourteen departments and about 10,000 students. During a private reception, great interest in Forth was expressed to us by students and representatives from all the university's departments. Afterward, we were led into an adjoining lecture hall, where a short FORML conference was held. In this room, only a small handful had used Forth or had seen it in action, but enthusiasm waxed strong. Some topics from previous meetings were re-introduced to the new audience, and fresh material was covered as well. Our visit to Xi'an, and to the People's Republic of China, ended with a friendly round of questions and answers, and an evening of comaraderie in the hotel lounge.

*The conference lectures in this article and in Part One (*Forth Dimensions *VI/5) were greatly abbreviated due to space considerations. Watch for published conference proceedings which may contain the papers on which these talks were based. Special thanks to Bob McGhee for helping to compile these notes during the trip.*

*— Marlin Ouverson.*

## PRODUCTS AND ANNOUNCEMENTS

Forth Dimensions *welcomes press releases and product announcements, as well as reader letters regarding product performance.*

Keep your calendars open, folks; **FIG members in West Germany are arranging a FORML conference** for October 1985. The technical content of the conference is anticipated to be excellent. Even so, organizers for the event have reserved a castle to entice us foreigners (but is it crenelated?). And if the conference happens to fall as close as one could wish to Oktoberfest, and just before a major European technical conference, then it must be coincidence. Call the FIG Hotline at 415-962-8653 for details.

Laboratory Microsystems, Inc. has released a **Unix implementation of Forth** for the IBM PC/AT. PC/FORTH for 80286 Xenix conforms to the Forth-83 Standard, and code written for other LMI systems can be ported to run under the company's latest product. Unix functions can be accessed from within this version of PC/FORTH, which supports the standard Unix file-and-directory interface, paths and logical devices. The full-screen editor which comes with the system is customized for the PC/AT; and this Forth can compile from text files generated with Unix's vi or ed, as well as from standard Forth screen files. LMI can be reached by calling 213-306-7412 (voice) or 213-306-3530 (300/1200 baud BBS for registered LMI customers).

MicroMotion has announced release of its **MasterForth for the Macintosh.** The system conforms to the Forth-83 Standard, includes a 68000 macro-assembler, and supports Mac's mouse, finder, menus and graphics toolbox. In addition to its line of Apple software, MicroMotion has versions of MasterForth for CP/M, the Commodore 64 and IBM PC. Floating point is an optional extension. Call MicroMotion at 213-821-4340 for further information.

Forth, Inc. informs us their **polyFORTH II is available.** It provides a multi-tasking, multi-user system to users of MS-DOS computers. Tasks may have private partitions or may execute shared, reentrant routines. The relatively new product is being used by Bell Canada to operate thirty-two terminals on an Omnibyte 68000 processor. Forth, Inc. may be contacted at 213-372-8493.

# YACS

*Henry Laxen*
*Berkeley, California*

One of the most frequent complaints newcomers to Forth mention is its lack of a **CASE** statement. I remember calling Kim Harris about eight years ago, when I was first learning Forth, to despondently ask how such an oversight was possible. He assured me that, while the language contained no intrinsic **CASE** statement, it would be little trouble to implement one. I am sure he and everyone else knowledgeable about Forth must have been asked that question at least a thousand times.

In order to lay the question to rest once and for all, *Forth Dimensions* devoted an entire issue to a case-statement contest. The fond hope was that the last word had finally been spoken on the subject of case statements. That was true, until now. During one of my recent Forth playfests, I wondered what would be the characteristics of the ideal case statement. As an exercise in Forthought, I would like to share with you my deliberations.

I started with the **CASE** known to many (see Figure One), which is probably the simplest of all. Basically, it defines an array of executable Forth words at compile time, then indexes into this array and executes the *n*th element at run time. Let's go through the definition of **CASE** in Figure One, word by word.

**CREATE** makes a header for us and adds it to the dictionary. It also initializes the code field of the newly defined word to be that of **VARIABLE. HIDE** then hides this definition (in F83) so that it is unavailable to the compiler until it is **REVEAL**ed later. **!CSP** is used to initialize the compile-time error checking. Next, **]** is called to invoke the Forth compiler. The counterparts to **HIDE** and **!CSP** will be executed when the definition is terminated by ;. Finally, the last word in **CASE** to be executed at compile time is the **(;CODE)** laid down by **DOES>**. It rewrites the code field of the newly defined word to point to the run-time code following it.

Now let's look at the run-time action. When **ROMAN** is called, it assumes there is an index on the stack. The run-time action of **DOES>** places the parameter field address of **ROMAN** on the parameter stack in front of the index. **SWAP** brings the index to the top of the stack. The **2*** doubles it, because each compiled pointer is two bytes long. **+** adds the doubled index to the origin of the array of pointers. **@** gets the compiled code field address and passes it to **EXECUTE**, which runs it.

This is the basic idea behind most Forth case statements in existence today. Let's look objectively at the advantages and disadvantages of this approach. The biggest advantage of the version described in Figure One is its simplicity. It would be quite difficult to make it simpler. It is also extremely fast. The amount of overhead required to decide which of *n* cases to execute is minimal, just a shift and an addition. On the minus side, it is very dangerous. If it is ever called with an undefined index value, chances are very good that we will crash. Speaking from experience, let me assure you this is not an infrequent occurrence.

The easiest way to solve this problem is illustrated in Figure Two. If we simply alter the syntax of a case statement by requiring that the number of cases precede the **CASE** defining word, we can simply and easily implement some run-time bounds checking. The word **MAP** will abort with an error message if the index is not between zero and number-of-cases minus one. Otherwise, it will return the address of the cell where the code field corresponding to that case is stored. We then fetch and execute that code field as before.

This essentially solves the problem of crashing case statements, but introduces another. First, we have changed the syntax of the statement in order to accommodate run-time checking. This would necessitate changing all occurrences of **CASE** in our source code. Furthermore, this form of case statement is difficult to maintain, since if we add or delete a case from the body of the statement, we must also remember to change the corresponding number at the beginning of the statement. This is a pain and often leads to bugs and frustration.

What we really want is the old syntax for a case statement (i.e., no preceding number), but run-time error checking as well. Greedy, aren't we? Fortunately, this is easy, too. The code listed in Figure Three performs this task. Let's look at how it works. After defining the header, we see the words **HERE 0 ,**. This phrase remembers the current dictionary location and reserves a word of storage there. We then proceed to compile the individual cases, just as before with **HIDE !CSP ]**. Once the ; is encountered, we execute the rest of the words between the **]** and the **DOES>**.

These words simply calculate the number of cells used for code fields and store the result in the cell reserved for it at the beginning of the definition. The phrase **HERE OVER -** calculates the number of bytes compiled since the original **HERE** and the terminating ;. The **2/** converts this byte count into a cell count. The **2-** is there because we don't want to count the cell used for this count ( **0 ,** ) or the cell used by the **UNNEST** compiled by ;. Finally, the **SWAP !** places the result of this calculation into the cell reserved for it at the beginning. Thus, we have computed the number of items in the array, and no longer require a count to precede the **CASE**. Notice that the run-time action of this statement is identical with the one in Figure Two.

We have progressed. We finally have a case statement with the advantages of the first one but without its major disadvantage. The cost in additional complexity is minimal — approximately three lines of code, counting the implementation of **MAP**. However, all these case statements lack one major feature, namely flexibility. The only cases we can select between are small integers. If we wanted to perform different actions depending upon the range of an integer, and if that range were very large, this approach would be quite unusable.

```
\ Simplest CASE Statement
: ZERO     ." Zero is not a Roman Numeral" ;
: ONE      ." I"   ;
: TWO      ." II"  ;
: THREE    ." III" ;
: FOUR     ." IV"  ;
: CASE
   CREATE   HIDE  !CSP  ]
   DOES>    SWAP 2* +   @ EXECUTE    ;
CASE ROMAN
  ZERO ONE TWO THREE FOUR    ;
0 ROMAN 1 ROMAN 2 ROMAN
3 ROMAN 4 ROMAN 5 ROMAN
```

Figure One

```
\ CASE Statement with Explicit Bounds Checking
: MAP   ( n addr -- addr' )
   2DUP @ U< IF   2+ SWAP 2* +
   ELSE   .S TRUE ABORT" Index out of Range"   THEN   ;
: CASE
   CONSTANT   HIDE   !CSP   ]
   DOES>   MAP  @ EXECUTE    ;

5 CASE ROMAN
  ZERO ONE TWO THREE FOUR    ;
0 ROMAN 1 ROMAN 2 ROMAN
3 ROMAN 4 ROMAN 5 ROMAN
```

Figure Two

```
\ CASE Statement with Implicit Bounds Checking
: CASE
   CREATE
    HERE 0 ,   HIDE !CSP ]
    HERE OVER -  2/  2-   SWAP !
  DOES>   MAP @ EXECUTE   ;

CASE ROMAN
  ZERO ONE TWO THREE FOUR    ;
0 ROMAN 1 ROMAN 2 ROMAN
3 ROMAN 4 ROMAN 5 ROMAN
```

Figure Three

Also, in order to use them we must define a word, and hence a new name, for each case. For me, this is an even greater disadvantage, since naming words appropriately is the hardest part of programming in Forth, the part I would like to minimize.

Thus, on the other end of the spectrum, I would be willing to sacrifice some speed and simplicity if I could have arbitrary selection criteria and avoid having to come up with a unique name for each of the cases. In addition, I would like this new case statement to introduce the least number of new words into Forth, ideally just **CASE** and, perhaps, **ENDCASE**. After thinking about this set of objectives for quite a while, I finally came up with an implementation. At this point, I challenge you to see if you can come up with one — I will reveal mine in my next article.

One closing comment. In Figure Three, where we implemented the case statement that calculates the number of items in the **CASE**, we made use of a property of ] that is relatively new and useful. In the days of fig-FORTH, ] was implemented by setting the variable **STATE** to a value that indicated we were compiling instead of interpreting. The word **INTERPRET** in the Forth system would look at this value of **STATE** and decide whether to compile or interpret the word it was scanning. Thus, ] was not callable as a subroutine, and the definition in Figure Three would not have worked. One of the big advances in Forth-83 was the discovery that a system could be written without the variable **STATE**. One consequence of this observation was that ] became a callable procedure, just like other Forth words. Its function was to compile the words in the input stream until a [ was encountered. (Note that ; calls [.) Thus, ] became a useful procedure in itself, allowing us to implement ideas such as those found in Figure Three. Needless to say, the specification for the **GRAND CASE** statement described in the previous paragraph makes heavy use of this feature of ]. Enough hints; good luck, and may the Forth be with you.

# Chapter News

We wish to welcome two new chapters:

Central Arkansas FIG Chapter, Little Rock, Arkansas

New Hampshire FIG Chapter, Manchester, New Hampshire

The two chapters in the San Francisco Bay area have decided to recombine as one for greater strength and companionship.

As we end this volume of *Forth Dimensions*, I want to thank the people who have put so much effort into making these sixty-nine chapters possible world-wide. To the "sparks," the people who work so hard getting the chapter started and keeping the day-to-day engine running; and to the members of these chapters, who give so much of their time to make Forth and the Forth Interest Group possible, I would like to say "Here, Here!" and "Thank You!" If I could speak for the rest of the Forth community, I think you would here 5000+ voices saying "Thank You!"

**Orange County FIG Chapter**

Nov. 28: Martin Tracy, from the Los Angeles Chapter, attended and presented his paper from FORML. Steve Pollack presented his first pass on a public-domain floating point for the 8087 on IBM PCs and clones. There was considerable discussion about the Forth Convention, and about what our chapter should be doing to promote Forth.

Dec. 5: We had Guy Kelley and Guy Grotke, guests from the San Diego Chapter. They presented a meta-compiler which can be taught easily. Guy Kelley also presented a paper on IBM PC key assignments, which can be vectored to do about anything. Bob Snook presented a paper on readability vs. economy.

— *Roland Koluvek*

**Netherlands FIG Chapter**
(in formation)

Some news from Holland: Forth is doing well here and is promoted by association with the Hobby Computer Club (HCC). The HCC is an organization of 20,000 members and still growing fast. It has a matrix structure; this means everybody is, at the same time, a member of some interest growp (row; e.g., Forth or Apple) and from a geographical subgroup (column; e.g., Utrecht or Amsterdam). Forth is spreading through contact with many BASIC programmers; we are showing off by winning the problem-of-the-month contest nearly every month; and we promote Forth in the HCC newsletter (also sold outside the HCC). Among activities are development of FYSFORTH, mentioned in *Forth Dimensions*. We have quarterly meetings and a bimonthly bulletin, *VIJGEBLAD*.

We would like to tell you about the availability of the 68000 fig-FORTH source listing written in Motorola assembler. It can be cross-compiled by the public-domain assembler available under CP/M. It assembles to an exact copy of the object of the existing 68000 fig-FORTH version 1.0. Plans are to make this source listing available in print in Holland and via FIG. We can send a machine-readable version. In Holland, it is also distributed on cassette in the Forth version of the BASICODE format. BASICODE is a joint effort of the HCC and the Dutch Broadcasting Foundation, "NOS." It is a method for distribution of machine-independent BASIC software. Every Sunday, a program is broadcast here and also on the "Wereldomroep" international shortwave. The 68000 FIG source was converted to this format with the cooperation of the Dutch HCC FIG chapter and the 68000 user group.

— *Albert van der Horst*

**Potomac FIG Chapter**

Feb. 5: Joel Shprentz described and demonstrated a version of Prolog in Forth. Prolog is a language used for artificial intelligence. Like expert systems, Prolog uses facts (e.g., Mary likes wine) and rules (John likes X if X likes wine) to make logical conclusions (John likes Mary). Joel has written a Prolog interpreter in Forth-83. Although the language syntax has been twisted to fit Forth, it is still Prolog (similar liberties have been taken with LISP implementations). The problem above would look like this:

```
ATOM John ATOM Mary ATOM likes
ATOM wine VAR X
{ { likes Mary wine } } RULE
{ { likes John X } { likes X wine } }
   RULE
{ { Likes John X } } PROVE
```

— *Joel Shprentz*

**Central Arkansas FIG Chapter**

Dec. 8: The Central Arkansas FIG Chapter was formed during its first meeting, held at the National Education Center's excellent facility. Thanks are in order — to the Arkansas Computer Club for support and assistance; to Betsy (FIG Hotline) for the great supply of handouts; to Ron and the Atlanta FIG Chapter for their wishes and support; and to John (SE Florida FIG) for the great input and screens. The meeting was called to order by Gary Smith, who gave a presentation of the history and architecture of Forth. Donald Long gave several informative demonstrations of Forth on a COMPAQ with LMI Forth. Donald also described the single-board Forth system he uses for home security. Gary demonstrated the stack using a Jupiter Ace running the stack/boxes demo.

— *Gary Smith*

### Kansas City FIG Chapter

Nov. 27: Thirteen attended. Brooks Herdon gave a demo of Harvard Software Forth. Some highlighted features included segmenting different functions of Forth, capability to load source from text files entered via Wordstar, a logging facility, and others. We also noticed that some words did stack-underflow/overflow checking.

### Sacramento FIG Chapter

Nov. 13: Twenty-three people attended, including Highlands High School teacher Jim Wong and two of his students, Trevor and John. They demonstrated their Forth-directed robot, TOPO. Mr. Wong purchased TOPO personally in order to enhance student interest in computer science with his proposed "Robotics and Forth" class, students' first exposure to Forth. Their impression is such that Mr. Wong is considering a Forth language class for the near future. We also had a round-table discussion regarding Charles Moore's talk on his Forth engine at the Silicon Valley FIG meeting, which several of our members had attended.

— *Tom Ghormley*

### Fort Wayne FIG Chapter

Nov. 14: Ed Harmon has selected the ACM Special Interest Group Graphics (SIGGRAPH) CORE standard as the task of our group project. He has chosen a Pascal implementation as a model to be implemented in Forth. Ed led the meeting discussion toward defining the task interface requirements with the following conclusions: 1) Forth-79 will be the dialect used. 2) The names and types of variables were selected. Some variables will be arrays and double precision. Floating point will not be used. 3) Ed partitioned the Pascal model's procedures and assigned them as tasks for the five members participating in the project.

— *Blair McDermid*

# FIG Chapters

## • ALASKA

**Kodiak Area Chapter**
Call Norman C. McIntosh
907/486-4843

## • ARIZONA

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

**Tucson Chapter**
Twice Monthly,
2nd & 4th Sun., 2 p.m.
Flexible Hybrid Systems
2030 E. Broadway #206
Call John C. Mead
602/323-9763

**Central Arkansas Chapter**
Twice Monthly: 2nd Sat., 2 p.m. &
4th Wed., 7 p.m.
Call Gary Smith
501/227-7817

## • CALIFORNIA

**Los Angeles Chapter**
Monthly, 4th Sat., 10 a.m.
Hawthorne Public Library
12700 S. Grevillea Ave.
Call Phillip Wasson
213/649-1428

**Monterey/Salinas Chapter**
Call Bud Devins
408/633-3253

**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

**Sacramento Chapter**
Monthly, 4th Wed., 7 p.m.
1798-59th St., Rm. A
Call Tom Ghormley
916/444-7775

**Bay Area Chapter**
Monthly, 4th Sat.
FORML: 10 a.m.
General: 1 p.m.
ABC Christian School Aud.
Dartmouth & San Carlos Ave.
San Carlos
Call: FIG Hotline — 415/962-8653

**Stockton Chapter**
Call Doug Dillon
209/931-2448

## • COLORADO

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

## • CONNECTICUT

**Central Connecticut Chapter**
Call Charles Krajewski
203/344-9996

## • FLORIDA

**Orlando Chapter**
Every two weeks, Wed., 8 p.m.
Call Herman B. Gibson
305/855-4790

**Miami**
Monthly, Thurs., p.m.
Coconut Grove area
Call John Forsberg
305/252-0108

**Tampa Bay Chapter**
Monthly, 1st Wed., p.m.
Call Terry McNay
813/725-1245

## • GEORGIA

**Atlanta Chapter**
Call Ron Skelton
404/393-8764

## • ILLINOIS

**Central Illinois Chapter**
Urbana
Call Sidney Bowhill
217/333-4150

**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., 10 a.m.
Call John Oglesby
317/353-3929

**Fort Wayne Chapter**
Monthly, 2nd Wed., 7 p.m.
Indiana/Purdue Univ. Campus
Rm. B71, Neff Hall
Call Blair MacDermid
219/749-2042

## • 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.
Wilbur E. Walker Co.
532 Market
Wichita, KS
Call Arne Flones
316/267-8852

## • LOUISIANA

**New Orleans Chapter**
Call Darryl C. Olivier
504/899-8933

## • MASSACHUSETTS

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

## • MICHIGAN

**Detroit Chapter**
Monthly, 4th Wed.
Call Tom Chrapkiewicz
313/562-8506

## • MINNESOTA

**MNFIG Chapter**
Even Month, 1st Mon., 7:30 p.m.
Odd Month, 1st Sat., 9:30 a.m.
Vincent Hall Univ. of MN
Minneapolis, 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 Tues., 7 p.m.
Thornhill Branch of
St. Louis County Library
Call David Doudna
314/867-4482

## • NEVADA

**Southern Nevada Chapter**
Call Gerald Hasty
702/452-3368

## • NEW HAMPSHIRE

**New Hampshire Chapter**
Monthly, 1st Mon., 6 p.m.
Armtec Industries
Shepard Dr., Grenier Field
Manchester
Call M. Peschke
603/774-7762

## • NEW MEXICO

**Albuquerque Chapter**
Monthly, 1st Thurs., 7:30 p.m.
Physics & Astronomy Bldg.
Univ. of New Mexico
Call Rick Granfield
505/296-8651

## • NEW YORK

**FIG, New York**
Monthly, 2nd Wed., 8 p.m.
Queens College
Call Ron Martinez
212/517-9429

**Rochester Chapter**
Bi-Monthly, 4th Sat., 2 p.m.
Hutchinson Hall
Univ. of Rochester
Call Thea Martin
716/235-0168

**Syracuse Chapter**
Monthly, 3rd Wed., 7 p.m.
Call Henry J. Fay
315/446-4600

## • OHIO

**Athens Chapter**
Call Isreal Urieli
614/594-3731

**Cleveland Chapter**
Call Gary Bergstrom
216/247-2492

**Cincinatti Chapter**
Call Douglas Bennett
513/831-0142

**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

**Central Oklahoma Chapter**
Monthly, 3rd Wed., 7:30 p.m.
Health Tech. Bldg., OSU Tech.
Call Larry Somers
2410 N.W. 49th
Oklahoma City, OK 73112

### • OREGON

**Greater Oregon Chapter**
Monthly, 2nd Sat., 1 p.m.
Tektronix Industrial Park
Bldg. 50, Beaverton
Call Tom Almy
503/692-2811

### • PENNSYLVANIA

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

### • TENNESSEE

**East Tennessee Chapter**
Monthly, 2nd Tue., 7:30 p.m.
Sci. Appl. Int'l. Corp., 8th Fl.
800 Oak Ridge Turnpike, Oak Ridge
Call Richard Secrist
615/693-7380

### • TEXAS

**Austin Chapter**
Contact Matt Lawrence
P.O. Box 180409
Austin, TX 78718

**Dallas/Ft. Worth
Metroplex Chapter**
Monthly, 4th Thurs., 7 p.m.
Call Chuck Durrett
214/245-1064

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

### • VERMONT

**Vermont 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

**First Forth of Hampton Roads**
Call William Edmonds
804/898-4099

**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.
154 Business School
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

### • BELGIUM

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

**Southern Belgium FIG Chapter**
Contact Jean-Marc Bertinchamps
Rue N. Monnom, 2
B-6290 Nalinnes
Belgium
071/213858

### • CANADA

**Nova Scotia Chapter**
Contact Howard Harawitz
227 Ridge Valley Rd.
Halifax, Nova Scotia B3P2E5
902/477-3665

**Southern Ontario Chapter**
Quarterly, 1st Sat., 2 p.m.
General Sciences Bldg.
Rm. 312
McMaster University
Contact Dr. N. Solntseff
Unit for Computer Science
McMaster University
Hamilton, Ontario L8S4K1
416/525-9140 ext. 3443

**Toronto FIG Chapter**
Contact John Clark Smith
P.O. Box 230, Station H
Toronto, ON M4C5J2

### • COLOMBIA

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

### • ENGLAND

**Forth Interest Group — U.K.**
Monthly, 1st Thurs.,
7p.m., Rm. 408
Polytechnic of South Bank
Borough Rd., London
Contact Keith Goldie-Morrison
Bradden Old Rectory
Towchester, Northamptonshire
NN12 8ED

### • FRANCE

**French Language Chapter**
Contact Jean-Daniel Dodin
77 Rue du Cagire
31100 Toulouse
(16-61)44-03

### • GERMANY

**Hamburg FIG Chapter**
Monthly, 4th Sat., 1500h
Contact Horst-Gunter Lynsche
Common Interface Alpha
Schanzenstrasse 27
2000 Hamburg 6

### • IRELAND

**Irish Chapter**
Contact Hugh Doggs
Newton School
Waterford
051/75757 or 051/74124

### • ITALY

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

### • REPUBLIC OF CHINA

**R.O.C.**
Contact Ching-Tang Tzeng
P.O. Box 28
Lung-Tan, Taiwan 325

### • SWITZERLAND

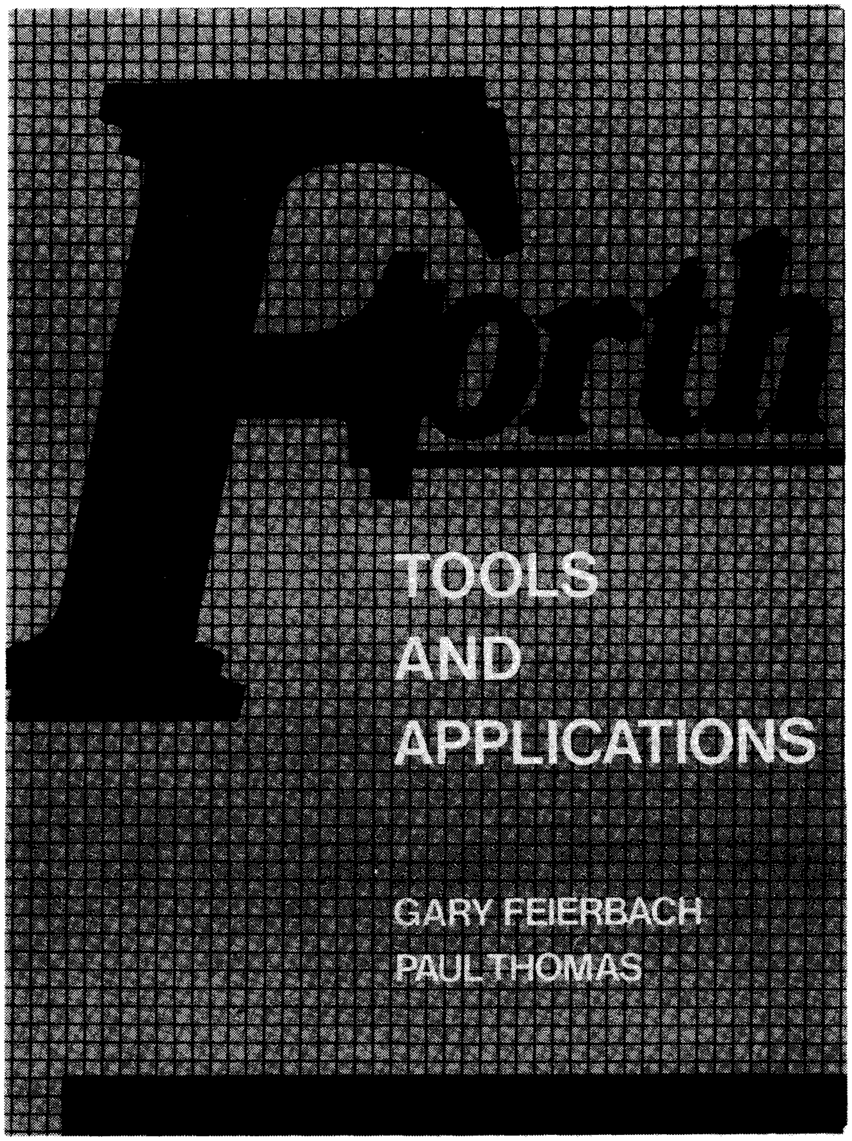
**Swiss Chapter**
Contact Max Hugelshofer
ERNI & Co., Elektro-Industrie
Stationsstrasse
8306 Bruttisellen
01/833-3333

## SPECIAL GROUPS

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

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

**FIGGRAPH**
Call Howard Pearlmutter
408/425-8700






**FORTH INTEREST GROUP**
P. O. Box 8231
San Jose, CA 95155

BULK RATE
U.S. POSTAGE
PAID
Permit No. 3107
San Jose, CA

ROBERT SMITH
2300 ST. FRANCIS DR.
PALO ALTO, CA 94303

Address Correction Requested