FORTH Dimensions

Volume 7, Number 4

November/December 1985

$4.00

Mailboxes & Multi-Tasking

Wordwrapping Tool

Benchmark Readability

Atari Graphics

F83 Word Usage

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.

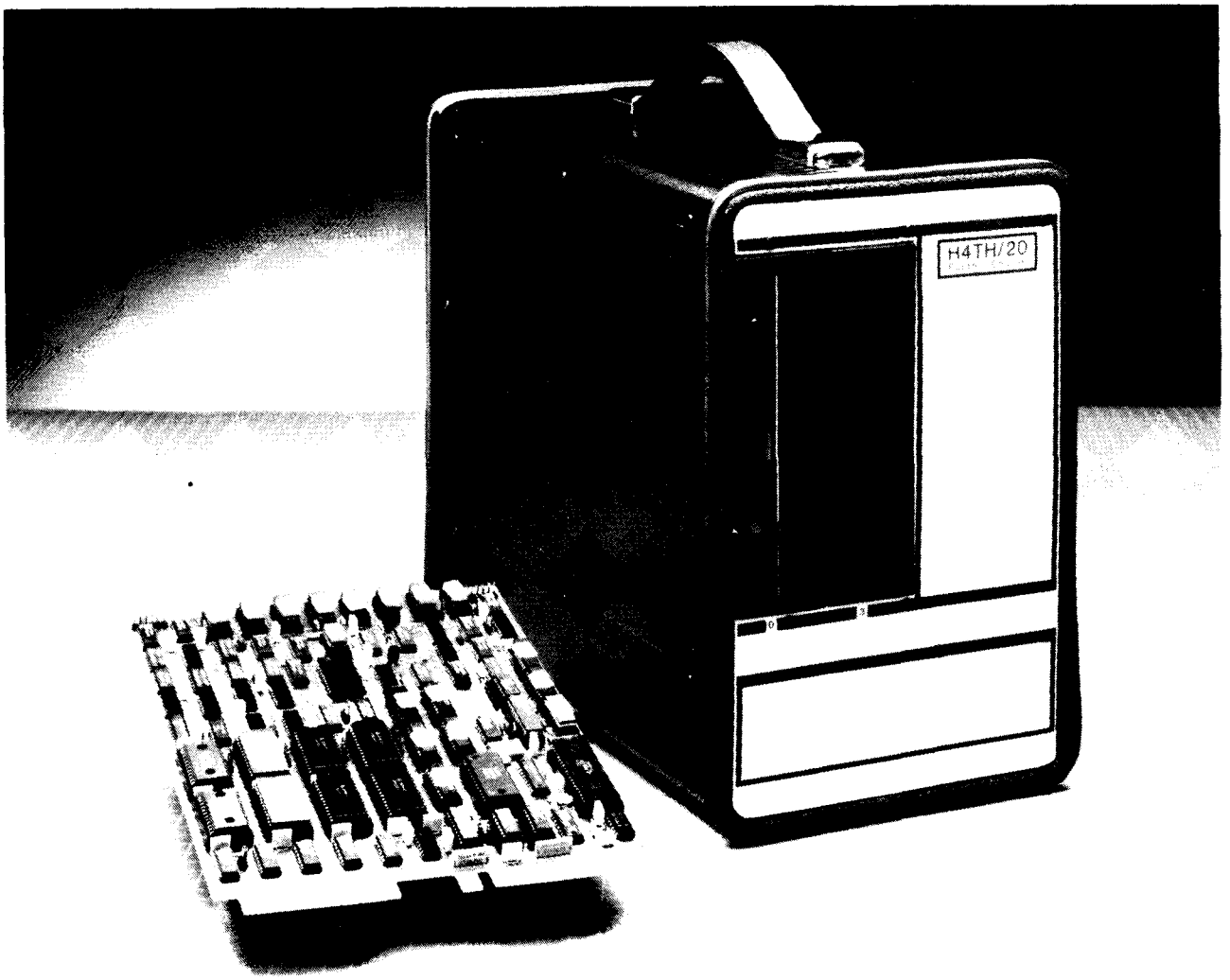
MICROPROGRAMMED BIT SLICE FORTH ENGINE
.Microcoded forth kernel
.Microcoded forth primitives
.Multi-level task switching architecture for real time applications
.Optional writable control store

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 VII, Number 4
November/December 1985

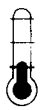
Editor
Marlin Ouverson

Production
Cynthia Lawson Berglund

Subscription to *Forth Dimensions* is free with membership in the Forth Interest Group at $20 per year ($33 foreign air). For membership, change of address and to submit material for publication, write to: Forth Interest Group, P.O. Box 8231, San Jose, California 95155.

ISSN No. 0884-0822

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

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:

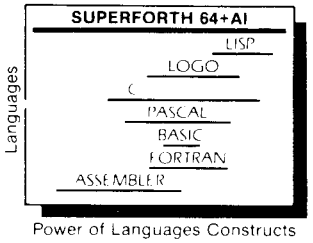
- Create Intelligent Programs
- Build Expert Systems
- Write Stand Alone License Free Programs
- Construct Rule Bases
- Do Knowledge Engineering
- Use Inference Engines

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

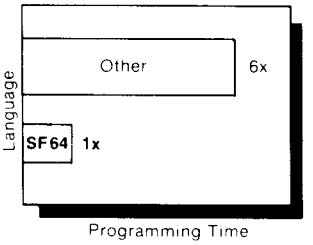
Extended Math Functions

- Fast ML Floating Point & Integer Math
- Double Precision 2E+38 with Auto. Sci Not.
- n^e^ 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

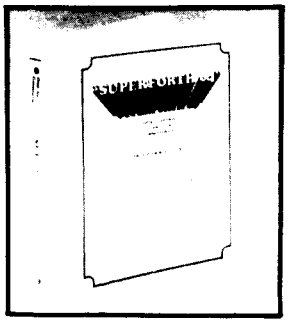
Utilities

- 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.

* Parsec Research
Commodore 64 TM of Commodore

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

Letters to the Editor

Multiply, Divide and Conquer

Dear Marlin,

I enjoyed Craig Lindley's Forth spreadsheet (*Forth Dimensions* VII/1, 2) with his application of Michael Stolowitz's algebraic parser (VI/6). He has aptly demonstrated how Forth concepts readily lend themselves to extensions.

However, the double number multiply and divide he offered on screen 30 are, in fact, mixed operators. Not only is the naming of these operators technically incorrect, but a limitation has also been introduced. The limitation is that mixed operators cannot be used in formulas to operate on data from two elements, since all data is stored as double numbers. *I.e.*, a[1 A * 2 a]a would fail.

I have enclosed code with double number operators. Using these operators will require that all numbers used in formulas will have to be expressed as double numbers. *E.g.*, a[3 A * 5.]a

Zafar Essak, M.D.
Vancouver, B.C.
Canada

In Praise of Libraries

Dear Editor,

I attended the recent FIG convention in Palo Alto. It was a wonderful chance to listen to professional Forth programmers, as well as to meet with others in various stages of understanding. I enjoyed meeting with many Forth celebrities: the energetic Ting, the pleasant Hall and the felt-hatted Moore.

There is one point, however, I feel must be raised. It seems to me that all over the world, Forth programmers are coding such things as: full-screen editors, turtle graphics, string words, floating point and extended addressing versions of Forth. While it is certainly an excellent learning experience to write such programs, why not make some of the best versions readily available to the public through FIG? The difficulty in obtaining such programs forces programmers to constantly "reinvent the wheel." Forth needs to progress continually. Once there is a common point for exchange of programs, people can concentrate on developing new applications, rather than reiterating old ones.

Lawrence Forsley, the convention banquet speaker, claimed the reason BASIC, Pascal and C have become so popular is that inexpensive versions were offered to schools and universities early. I envision a full-function package for Forth to rival GWBASIC, Turbo Pascal and C. Why not? Forth is a better language than any of the others.

This package would come complete with a full-screen editor, floating-point support, graphics, music and programming tools such as a cross-reference utility (how many people out there know such a program already exists in the public domain?). This Forth would use extended addressing (on PCs) to give the programmer the use of all available memory without the corresponding loss of speed associated with thirty-two-bit Forths on sixteen-bit machines. Some people at the F83 meeting asked to see a Forth program that took more than 64K. My reply is that once you load the full-screen editor, the programming tools, turtle graphics and floating point (preferably done on a math chip like the 8087), there is little room for programs. Indeed, I have already had trouble with overwriting the system that way. Some of these words, like the editor vocabulary, could reside in another segment and pop down only when necessary.

This package, or library, would be developed in the public domain, therefore satisfying universities' need for software in microcomputer labs that they can offer freely to students. (Many are doing this with programs such as PC Write and PC File.) Since Forth-83 is the new standard, and because it is such a clean implementation, FIG should adopt Laxen and Perry's F83 as the main version for this package. F83 is also well documented and so is perfect for students. Such an investment in education would yield

```
( DM* D*                Double number       850805z) DECIMAL

: DM* ( d1,u--dprod)
    DUP ROT * ROT ROT UM* ROT + ;

: D* ( d1,d2--dprod)
    2DUP 0 0 D< >R DABS
    DABS >R >R 2DUP R> DM*
    R> ROT ROT >R >R DM* 32768 DM* 2 DM*
    R> R> D+ R> IF DNEGATE THEN ;
```

```
( DM/MOD D/ DMOD        Double number       850805z)
: DM/MOD ( d1,u--drem,dquot)
    SWAP OVER /MOD >R SWAP UM/MOD >R 0 R> R> ;

: D/ ( d1,d2--drem,dquot) 2DUP 0 0 D< >R
    2DUP DABS SWAP DROP
      IF R@
        IF >R >R DNEGATE R> R> DNEGATE
        THEN SWAP >R /MOD R> SWAP >R
          0 R@ S->D D* D- R> S->D
      ELSE R@
        IF >R >R DNEGATE R> R>
        THEN DROP ABS DM/MOD
      THEN R> IF >R >R DNEGATE R> R> THEN ;

: DMOD ( d1,d2--drem) D/ 2DROP ;
```

high rewards in the use and acceptance of Forth in the future; it would benefit Forth programmers everywhere. Even the vendors would benefit, because as students found the need for the increased speed of Forths implemented in assembler, as well as target compilers, they would seek out FORTH Inc., or Laboratory Microsystems or one of the other vendor-supported Forths.

One interesting idea discussed at the meeting was the possibility of a FIG bulletin board to distribute Forth programs. This is an excellent idea which should be implemented as soon as possible. Yet, many programmers have no access to modems and could not benefit. Why not add to FIG's mail order service? BASIC programs (as well as programs in some other languages) are distributed by organizations like PC-SIG. Why should Forth lag behind? Let other programmers know how good Forth is — display your quality programs!

Forthfully yours,

Mark Smiley
Montgomery, Alabama

A CASE of Pairs

Dear Marlin,

Despite the valiant efforts of Henry Laxen ("YACS," *Forth Dimensions* VI/6 and VII/1), the case of **CASE** by Dr. Charles Eaker achieved what you may call a "semi-standard" status. Recent evidence is provided by the "Forth Spreadsheet" by Craig A. Lindley. Besides, many Forth implementations include the earlier **CASE** among their system extensions. In particular, my TI Forth system provides such an example. But quite often one would like to have a case statement which is more powerful than the standard usage of Dr. Eaker's **CASE**. And for many reasons, I do not like unnecessary expansions of my system. But the situation is far from being hopeless.

Let me show that the old **CASE** is much more flexible than it appears at first glance. The point is that **OF** expects on the stack a pair of numbers. If they are the same, then both will be consumed and a whole clause between **OF** and **ENDOF** will be executed, followed by **ENDCASE**. Otherwise, the top value will be dropped and execution will continue with the statement past **ENDOF**. Nothing prevents us from supplying the numeric pair for **OF** during run time. This simple remark extends enormously a range of applications. Using the above as a guiding principle, let's rewrite an example provided in "YACS, Part Two" (VII/1, p. 39). Please refer to the code in figure one.

Actually, for range classifications like the one presented here, I prefer a slightly different approach. Instead of checking an original condition, one may use a negation of it. So the flag will be zero if the original one was satisfied, and some non-zero value otherwise. Therefore, **OVER +** will provide a numeric pair suitable for **OF**. See figure two for a demonstration of this.

The demonstrated technique, though implicit in the definition, does not seem to be very well known.

Sincerely yours,

Michael Jaegermann
Edmonton, Alberta
Canada

```
HEX
1 CONSTANT TRUE           \ system dependent
: BELOW1 SWAP DROP        \ drop flag from the previous test
      OVER  > TRUE   ;    ( n flag limit -- n flag1 flag2 )
: CLASSIFY1 ( byte -- )
         0 CASE           \ leave dummy flag on stack; anything will do
    20 BELOW1 OF ." Control character"   ENDOF
    30 BELOW1 OF ." Punctuation"         ENDOF
    3A BELOW1 OF ." Digit"               ENDOF
    41 BELOW1 OF ." Punctuation"         ENDOF
    5B BELOW1 OF ." Upper Case Letter"   ENDOF
    61 BELOW1 OF ." Punctuation"         ENDOF
    7B BELOW1 OF ." Lower Case Letter"   ENDOF
    7F BELOW1 OF ." Punctuation"         ENDOF
    80 BELOW1 OF ." Control character"   ENDOF
                 ." Not ASCII character"
         ENDCASE DROP ;  \ drop an original value; unused
                         \ flag removed by ENDCASE
```

Figure One

```
: BELOW2  ( n limit -- n m ) OVER 1+ < OVER + ;
: CLASSIFY2 ( byte -- )
         CASE
    20 BELOW2 OF ." Control character"   ENDOF
    30 BELOW2 OF ." Punctuation"         ENDOF
    3A BELOW2 OF ." Digit"               ENDOF
    41 BELOW2 OF ." Punctuation"         ENDOF
    5B BELOW2 OF ." Upper Case Letter"   ENDOF
    61 BELOW2 OF ." Punctuation"         ENDOF
    7B BELOW2 OF ." Lower Case Letter"   ENDOF
    7F BELOW2 OF ." Punctuation"         ENDOF
    80 < TRUE OF ." Control character"   ENDOF
                 ." Not ASCII character"
         ENDCASE ;
```

Figure Two

Errata

In our last issue (VII/3), the illustrations for figures one and two on page thirteen were accidentally switched. Our apologies go to Professor Yngve and to readers who may have been confused by the error. Also, as author Schmauch points out in his "Pseudo-Interrupts" article in the same issue, his code is in Forth-79 and not Forth-83, as it was labelled. —Ed.

Leaders' Edge

We recently received a call from Dennis Wilson, the local FIG Chapter Coordinator in Phoenix, Arizona. He asked if we would communicate to all concerned that Charles Moore, the inventor of Forth, is scheduled to speak at a dinner and conference at the Arizona Biltmore Resort on December 6th. If you plan to be in the area and would like more information, call Dennis at 602-957-0469.

It has become apparent that, at least to a few, there is still some confusion between the Forth-83 Standard and F83. For those who aren't absolutely clear as to the distinction, here goes: Forth-83 is the set of standard Forth words arrived at by the Forth Standards Team. That team is composed of various vendor representatives, systems and application programmers and, generally, Forth experts. Their opus became the fundamental instruction set that is Forth-83 and was intended as an improvement over Forth-79 and the earlier fig-FORTH. This word set had to be defined at a level of generality which covers all computers, and thus could not include anything specific to any single piece of hardware. It is comprised of specific operators and does not touch on, for example, extensions such as an editor, graphics or even cursor positioning; it does include all the principle operators and what their effects must be, but not how they should work internally.

F83, by contrast, is the name of a public-domain implementation based on Forth-83 and was developed by Henry Laxen and Michael Perry. It also contains a number of non-standard words and extensions, but no support. Whether these make F83 usefully complete with tools and source code, or overly complex and memory intensive, appears to be a matter of personal taste and system constraints. Some people, unfortunately, have come to believe that F83 is the only implementation of the Forth-83 Standard. This is not the case, as there are systems available commercially, with full vendor support, that comply with Forth-83 (e.g., Laboratory Microsystems and MicroMotion).

When it gets down to the nitty-gritty, often there are logical choices about how to implement *any* language — a standard only requires that the words it contains behave in standard ways, but does not address how an implementor makes them perform in that manner. The differences among implementations are in part the natural result of developers making decisions based on their own priorities, convictions and techniques. Along with documentation, support and price, the result is healthy competition between vendors in the growing Forth market. Which, in turn, should result in continually improving products and services.

The Forth Interest Group has not endorsed any Forth implementation since those old days when the only way to get the language onto a microcomputer was to order a fig-FORTH listing and install it manually. Nor has FIG or the Forth Standards Team yet undertaken to certify systems as conforming, or not, to the standard. But with diligent comparison of the Forth-83 Standard document (available on the FIG order form) and a particular Forth system's documentation, and by using vendor-provided support channels, an educated consumer can determine whether an implementation operates in accordance with the standard, and can estimate closely whether it will meet the requirements of a particular operation or project.

There are probably fifty or more Forth vendors, of which about a half dozen are leaders in the field. They represent a wide variety of business and programming practices which serve the diverse needs of the marketplace. And how that marketplace has changed over the years! Forth may have been something of a novelty in the beginning, appealing mostly to language buffs and people trying to hack their way out of pure assembly code, but evolution continued and Forth slipped through the doors of the Fortune 500. Now it appears that professionals are in the majority, using Forth in corporations that sometimes treat it like a trade secret, a hidden advantage over their competitors. We find Forth at IBM, AT&T, Kodak, Hewlett-Packard and Lockheed, among many other notable companies.

Sometimes we all need to step back and get a large view of where we stand in relation to the world around us. This is a time of opportunity in which almost anything can happen. Bill Ragsdale, FIG's founding president, often speaks of leadership and its importance to vendors' success. That trait applies at the level of the individual as well. And what is a leader but someone with a vision of the broad scheme of things, the flexibility to adapt to changing conditions without becoming dogmatic, the imagination to see the hidden potential in anything and the courage needed to define and achieve specific goals?

Think of the Forth Interest Group as an association of actual and potential leaders. The possibilities are immense!

—Marlin Ouverson
Editor

Application Tutorial

Wordwrapping Tool

Michael Ham
Santa Cruz, California

Forth vendors must steer a course between the Scylla of an absolutely minimal system and the Charybdis of a full-blown maximal system. In the minimal system, the Forth contains as little beyond the standard requirements as is necessary to have a system that runs at all. The amount of free memory is enormous, but the user must create the entire range of developmental tools. The maximal system, on the other hand, includes every conceivable tool a user could want, but available memory shrinks to the size of a pea.

Most vendors approach the solution by including in the core system the factored-out essentials for a variety of useful tools, and providing as optional extensions or overlays those more comprehensive facilities likely to be useful in some, but not all, applications. Double-precision and floating-point operations, for example, are often provided as separate files; a screen editor and graphics operators might be supplied as precompiled overlays, loaded only when needed.

In my current project, I am using PC/Forth (Laboratory Microsystems, Inc.). As the name implies, this is a Forth specifically designed to exploit the powers and peculiarities of the IBM PC and its clones. To that end, PC/Forth provides, in addition to the Forth-83 Standard word set, various tools that make the developer's life easier in this particular environment.

Some of these are complete overlays, such as the DOS interface, which allows the developer to read and write regular DOS files, in addition to the words that permit the usual Forth operations (**BLOCK, FLUSH, UPDATE,** etc.) on DOS-formatted screen files. Others are proto-tools: for example, variables that can be used in the creation of various helpful tools. The variable **COMPAQ**, for example, gives the program a way to know whether the computer is a Compaq or not, which can be useful in controlling the screen display.

The variable **OUT** in PC/Forth tells you how far you have gone in the output line. This is useful in various situations. For example, you might type something whose length you cannot predict, but from which you need to tab to another specific location. PC/Forth's word **.FILENAME**, for example, prints the name of a file. In PC/MS-DOS, however, the length of a filename can vary considerably, particularly when (in PC-DOS 2.x and later) the name printed by **.FILENAME** includes a long string of directories and subdirectories: FORTH DEVEL EA MAIN.SCR is a not-too-fanciful example.

By using **OUT**, however, you can readily create a word that will tab to a specific position on the line, even after output of unpredictable length:

```
: TAB  ( n --- ) ( tabs to designated
  position )
  OUT @ - SPACES ;
```

(The PC/Forth **SPACES** includes, in effect, a **0 MAX** so that negative arguments produce no spaces.) **OUT** is incremented by all the output words (**EMIT, TYPE, SPACE** and **SPACES**), updated whenever the cursor is repositioned, and zeroed by **CR**. The user can alter as well as examine **OUT**, should that prove useful in some situation. You can readily define **OUT** in other Forths by redefining the appropriate words:

```
VARIABLE OUT
  : TYPE  ( a #---) DUP OUT +! TYPE ;
  : CR  OUT OFF CR ;
```

etc. The definitions in PC/Forth are, of course, written in code for speed.

Here is a slightly more ambitious application of the use of **OUT**: **CAREFULTYPE**, a version of **TYPE** that observes a specified margin, does not break words, and indents each line a specified amount. The definition uses another vendor-supplied tool word, **SCAN**.

SCAN searches for the first occurrence of a given character in a string (in this case **BL**, a constant equal to the ASCII value for a blank). Given the address and the length of a string, **SCAN** searchs for the first instance of a specified character and returns the address of the character and the remaining length of the string. If the character is not found, the address returned is of the byte immediately following the string and the length returned is zero.

Two different high-level definitions of **SCAN** are shown. (PC/Forth's **SCAN** is in code for compactness and speed.) One definition uses a countdown loop based on the length as index; the other definition uses an indefinite loop and avoids the use of a variable at the cost of some stack-thrashing. (Because PC/Forth is 83-Standard, the **LEAVE** in the first definition jumps to the end of the loop; this action can be mimicked in 79-Standard systems with an **IF ELSE THEN** construct.)

In writing a definition dealing with strings, most of us aren't sure whether a given index is okay as is, or whether it should be bigger or smaller by 1. In writing the first definition of **SCAN**, for example, I didn't know offhand whether the value left on the stack should be 1, or 1 less than 1, or 1 greater. The nice thing about Forth is that such questions can be more quickly and more easily (and, for me at least, more accurately) answered through experiment than through analysis. The use of experiment is especially straightforward in the case of **SCAN**, which was written to mimic an existing word. The same method also worked in defining **CAREFULTYPE** to arrive at the 1- in lines 9 and 13 and the subtraction in line 10 (where I didn't know at first whether the difference itself was the number I wanted, or whether I needed to adjust it by 1 in one or direction or the other). In both words, in fact, my initial guesses were wrong, but I readily corrected the definitions on the spot.

Which definition is faster? Such questions are again best resolved through a quick experiment. A couple of test loops will provide the answer for your own system.

The **WHILE** clause in **CAREFULTYPE** contains the actions needed to set the loop up for another repetition (in this case, the action is to increment the address to point **SCAN** at the address of the byte immediately following the blank it found the previous time).

Because **SCAN** returns the bytes remaining in the string, the 1- in line 10 of **CAREFULTYPE**'s definition decrements the bytes remaining, thus directing **SCAN**'s attention one position to the right. **CKMARGIN** uses **OUT** to determine where we are, and to start a new line if the current word would have taken us past the right margin. The various stack manipulations jockey the addresses and the counts to the appropriate positions. Because **SCAN** finds each blank, a series of multiple blanks in the input are faithfully reproduced in the output, as shown by the demonstration.

The word " used in **TESTSTR** of the screen is delimited by the following " and stores the defined string in the dictionary, preceded by a count byte. When the word **TESTSTR** is executed, the address of the count byte is left on the stack. Other Forths have other ways of defining strings; for this demonstration, you need only find some way to put the address and count on the stack.

Any particular user, of course, will always want the vendor to steer a bit closer to Charybdis and include the special feature the user wants. For example, I would like to have a word **LEFT?** that would put a true flag on the stack if the last loop were exited by **LEAVE** and a false flag if the loop ran to completion. **LEFT?** would obviate the need for such variable switches as seen in the definition of **SCAN**. But vendors certainly know by now that users — particularly Forth users — are never satisfied. Forth's particular strength is that dissatisfied users can seek satisfaction by adding their own commands to the language.

```
0: ( High-level SCAN, definition 1          Ham  11.15 09/12/85 )
1:
2:  VARIABLE TSW  ( F if loop exited via LEAVE )
3:
4: : SCAN  ( adr len char --- char-adr len-remaining )
5:   TSW ON
6:   0 ROT DO OVER C@ OVER = ( same as char? )
7:       IF DROP ( char ) I ( len remaining ) TSW OFF LEAVE
8:       THEN SWAP 1+ SWAP  ( incr address )
9:       -1 +LOOP  ( decr length remaining )
10:   TSW @ IF DROP ( char ) 0 ( not found ) THEN ;
11:


0: ( High-level SCAN, definition 2          RGD 23:49 03/04/85 )
1:
2: ( adr len char --- adr' len' )
3:
4: : SCAN     >R 0                    ( adr len count )
5:          BEGIN  2DUP <>           ( adr len count flag )
6:               3 PICK C@    ( inspect char from string )
7:               R@ <>             ( check for match )
8:               AND             ( or string exhausted )
9:          WHILE  1+ ROT 1+ ROT ROT   ( no, inc count&adr )
10:         REPEAT  R> DROP - ;        ( return adr' len' )
11:


0: ( String printing word                  Ham 11:11 09/11/85 )
1:
2:  5 CONSTANT INDENT    ( amount each line indented )
3:  70 CONSTANT RTMARGIN
4:
5: : CKMARGIN ( # -) OUT @ + RTMARGIN > IF CR INDENT SPACES THEN ;
6:
7: : CAREFULTYPE ( adr # - \ adr=1st char of string; #=its length )
8:  BEGIN 2DUP BL SCAN ( leaves adr and amount of string left )
9:    DUP IF ( inside string ) 1- ( move past the blank ) THEN
10:    ROT OVER -           ( gives count of word+blank )
11:    DUP CKMARGIN          ( CR if needed )
12:    3 ROLL  SWAP TYPE  ?DUP ( any string left? )
13:  WHILE  SWAP 1+ SWAP      ( move adr past the blank )
14:  REPEAT  DROP ( adr ) ;
15:

0: ( Demonstration of CAREFULTYPE          Ham 16:08 09/11/85 )
1:
2: : TESTSTR " This is a string long enough to wrap around.  Don't
3: forget that strings longer than 256 need special treatment; the
4: count here is limited to a     byte." ;
5:
6: TESTSTR COUNT ( produces correct adr & count ) CAREFULTYPE
7:
```

Word Indexer

Mike Elola
San Jose, California

Users of programming languages often make use of programming aids. One such aid automatically locates all ocurrences of a specified variable throughout a program. Since Forth variables are not very different from Forth procedure calls, just one utility could be used to perform two functions: locate references to a target word within all other application words, and locate references to a target variable within application words. The words **CALLOUT** and **CALLOUTS** provide functions similar to these for the Forth language. They are defined in the three accompanying screens and adhere to the Forth-79 Standard, so should work "as is" on most Forth systems.

The uses I have found for these words are numerous; many were not even anticipated. If you wish to locate all the words which make use of a non-standard Forth word, you can do so. If you wish to see any of the places you may have used slow or tricky words like **DEPTH** or **PICK**, you can do so. If you would like an example of the usage of an unfamiliar word, you can query the words already in the dictionary. And even if you would just like to change the name of a word, you can locate the places where corresponding changes are required.

Originally, I wanted to see how the effects of a substantial change to a low-level word would ripple through a completed application. I needed to know how many other words would be impacted by the change. Rather than search for ocurrences of the word in listings of my application, I used **CALLOUT**. You might want to use these aids for still other purposes, e.g., streamlining applications and enhancing documentation.

CALLOUTS can help you to break down an application into discrete componenets. Leo Brodie introduces the concept of components in his recently published *Thinking Forth*. Documenting the components in your application can be a valuable exercise, especially if you want to be able to make changes later.

The elective words not referenced by your application are good candidates for removal. **CALLOUTS** shows these words as well as those that are referenced. It can also show where elective words are used in other elective words. While you might know that a particular elective is being used, you may not know the other electives it may rely upon.

Even the words in the non-elective part of Forth can be analyzed with these utilities. This provides you with additional documentation regarding an implementation of the Forth language. Once you know the components within your implementation of Forth, you can more deliberately avoid or engage them in your own program.

Those components you cannot completely avoid, you can prepare to forget. Identify the words you wish to preserve and include definitions of them at the start of the new application. Then you can safely "forget" the larger component from which they came.

How It Works

Word cross-references are found by searching a certain portion of the Forth dictionary for the code field address (CFA) of the query word, or current query word. The search starts at the top of the dictionary and proceeds down to the query word itself, so that recursion is detected. In the case of **CALLOUT**, the query word is the word you specify. However, with **CALLOUTS**, the query words are those starting with the most recent dictionary entry and proceeding down to the dictionary word that you specify.

Floor Enhancement

Normal operation of **CALLOUT** and **CALLOUTS** can be modified through the use of **FLOOR**. **FLOOR** allows specification of an alternate search limit, rather than the query word itself. You specify the new search limit with an existing word. Thereafter, any references to the current query word are only recognized if made from the part of the dictionary above the "floor" word. Often, response time improves dramatically when you specify a floor word.

Such a search limit is needed before you can request a report showing all the words which are directly referenced in your application. For example, if the first word loaded in your application is **START**, use the following commands:

```
FLOOR START ok
CALLOUTS !
CALLOUTS .....
CALLOUT .....
<CALLOUT>...CALLOUTS CALLOUT
FLOOR .....
etc.
```

Assuming ! is the first word in the dictionary, all the words in the dictionary eventually become query words, but only those instances of the query words that occur within **START**, and more recent words, will be reported.

A similar report is obtained by using the first of any elective words as the argument to **CALLOUTS**, after you have set **FLOOR** as described above. Once **CALLOUTS** reaches the elective words, it reports them only if they are directly referenced from your application. (References to them that are not directly made from your application are ignored.)

To deactivate **FLOOR**, use **FLOOR** with ! as its argument. The floor search limit is only active when it is higher than the current query word. So, the request

FLOOR ! (presumably) sets a minimum floor value that no longer affects the functions of **CALLOUTS** and **CALLOUT**.

To sum up, the utilities presented in this article allow you to easily find the interrelationships among dictionary words. Occurences of a given word in the definitions of other dictionary words can be found with **CALLOUT**. By using **CALLOUTS** with the **FLOOR** option, almost any subset of dictionary words can be located within other subsets of dictionary words — the most useful of which are listed following:

- application words in context of other application words
- application words against the entire dictionary
- application words against themselves and the electives only
- elective words in context of other elective words

```
SCREEN #60
 0) ( FLOOR*  PREVNFA  UPNFA  UMAX  DOTS  LEADID.  )
 1)   VARIABLE FLOOR' VARIABLE UPNFA
 2) : PREVNFA   ( HERE/NFA -- PREVNFA )
 3)    DUP HERE = IF DROP LATEST ELSE   PFA 4 - @ THEN ;
 4) : UMAX DDUP U< IF SWAP THEN DROP ;
 5)
 6) : FLOOR    ( <WORD>  (  -- )
 7)    -FIND 0= IF CR ." BAD WORD" ABORT
 8)    THEN DROP  FLOOR' !  ;  FLOOR !
 9)
10) : DOTS      ( COUNT -- )
11)    0 DO 46 EMIT LOOP ;
12)
13) : LEADID.          ( NFA -- NFA )
14)    DUP ID. 20 OVER C@ 31 AND -
15)    4 MAX DOTS ;


SCREEN #61
 0) ( <CALLOUT>                  )
 1) : <CALLOUT>           (  CFA -- )
 2)    HERE UPNFA !
 3)    BEGIN
 4)     UPNFA @ DUP PREVNFA PFA DO ( CFA -- )
 5)      DUP I 2- @ DUP [ ' <."> CFA ] LITERAL =
 6)      IF ( SKIP STRING DATA )
 7)         I DUP C@ + 1+ R> DROP >R
 8)      THEN [ ' <LIT> CFA ] LITERAL =
 9)      IF ( TEST FOR PFA ) 2+ THEN
10)      I @ =  IF ( EUREKA !!)
11)       UPNFA @ PREVNFA ID. 2 SPACES
12)       LEAVE THEN  ( CFA --)
13)     LOOP UPNFA DUP @ PREVNFA SWAP !
14)     UPNFA @ OVER FLOOR' @ UMAX U<
15)    UNTIL ( CFA -- ) DROP  ;


SCREEN #62
 0) ( CALLOUT  CALLOUTS   )
 1)
 2) : CALLOUT  ( <WORD>  (  -- )
 3)   -FIND 0= IF CR ." BAD WORD" ABORT
 4)    THEN CR DROP DUP NFA
 5)    LEADID. DROP CFA <CALLOUT> ;
 6)
 7) : CALLOUTS    ( <BOTTOM.WORD> (  -- )
 8)    -FIND 0= IF CR ." BAD WORD" ABORT
 9)    THEN DROP CR CR NFA PREVNFA HERE
10)    BEGIN ( BOTTOM.NFA NFA -- )
11)      PREVNFA  DDUP U< WHILE
12)      LEADID.
13)      DUP PFA CFA <CALLOUT> CR
14)    REPEAT  DDROP ;
15)
```

F83 Word Usage

C.H. Ting
San Mateo, California

How often various Forth words are used is a question interesting to most Forth programmers because this type of information can lead to better design and to the optimization of Forth systems. Most frequently used words should be coded in machine language for execution speed. They should also be at the top of the dictionary to minimize the time for interpretation and compilation.

A number of years ago, Don Colburn mentioned at a FORML meeting in Hayward, California that he used an extra cell in a word's header to accumulate statistics of word usage, either during compilation or during execution. He also mentioned that the most often used Forth word was (for comments, which was rather unexpected at the time. Since I haven't the luxury of metacompiling my own system with that type of flexibility, this concept remained a distant curiosity for me.

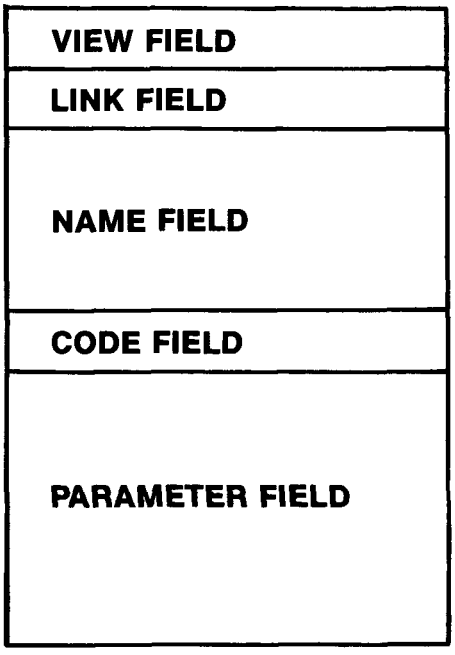


Figure One-Forth word layout in F83

After plunging into the F83 system produced by Mike Perry and Henry Laxen, I found a ready solution to analyze Forth word usage without much hard work. The secret is in the extra cell used in F83 to store the "view file" information. As shown in figure one, Forth words in F83 are laid out in the dictionary with five fields.

The view field stores a file number in its upper four-bit subfield and a block number in the lower twelve-bit subfield, allowing the source screen containing the word definitions to be retrieved from the disk and viewed by the user. If I am not going to use the view field for viewing purposes, I am free to use it for whatever I wish to do with it. Why not use it to accumulate the statistics of Forth word usage?

To use the view field for this purpose, I must do it in the following sequence:

1. Clear the view fields of all words in the dictionary.
2. Build a word processor which will scan a screen of code and increment the view field when the corresponding word is found in the screen.
3. Tabulate the statistics.

The program shown in the accompanying screens performs these functions. It looks extremely simple because it utilizes many powerful and interesting F83 features which require some explanation.

The most important feature I make use of is the vectored execution procedures, which allow me to assign a number of tasks to a single word. For example, I want to scan the dictionary and clear all the view fields before analyzing word usage. After the statistics are collected, I want to scan the dictionary and print the contents of the view fields. The scanning operations in both cases are identical. The difference is the actions I have to take after I find a view field. Anticipating that different actions are to be taken, I defined a vectored word **WORK** as a **DEFER**red word, and use it in the definition of the scanning word **WORKS**, which follows the dictionary link to locate every word in the dictionary and perform **WORK** on each of them.

WORKS is complicated by the fact that F83 hashes the dictionary linkage into four threads, and all four threads have to be scanned when traversing the dictionary. The definition is very similar to the word **DEFINED** which does the dictionary search for the text interpreter in F83. It scans all four threads and processes the one with the highest link field address. The process continues until all four link addresses are reduced to zero, indicating the end of the threads. The scanning is performed only on the **FORTH** vocabulary, in which all the primitive Forth words reside. (Usage of words in other vocabularies is much less frequent and the statistics are less significant than those of the words in the Forth vocabulary.)

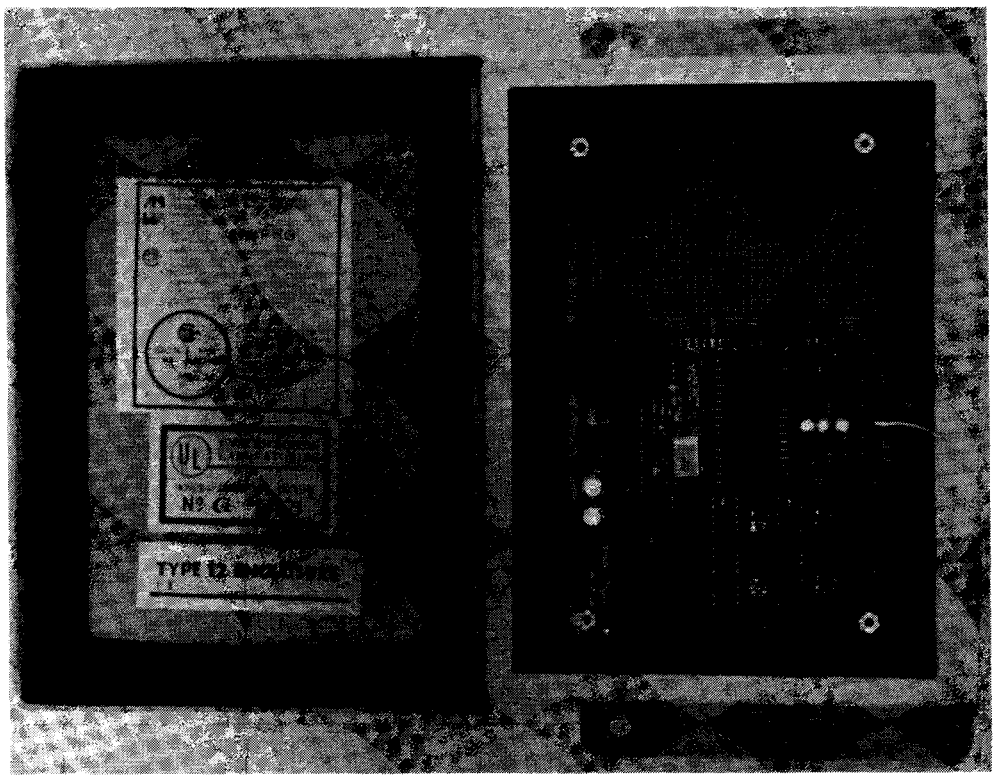
After **INIT-VIEW** is defined to clear the view field, given a link field address, we can zero the view field of all the words in the **FORTH** vocabulary by vectoring **WORK** to **INIT-VIEW** and executing **WORKS**. After the statistics are accumulated in all the view fields, we will vector **WORK** to **PRINT-VIEW** and then execute **WORKS**. This time, **WORKS** will print the contents of the view fields with the corresponding word names.

ACCUMULATE in screen 19 is the word processor which processes source screens very much like **INTERPRET**. If a word is found in the dictionary by **DEFINED**, the view field of this word is incremented. If a word is not found in the dictionary (actually, in the **FORTH** vocabulary), it is simply skipped. I couldn't care less if it is a number, which will be ignored also, unless it is **0**, **1**, **2** or **3**, which are Forth words.

In F83, **LOAD** is also a vectored word. I define **[LOAD]** to use **ACCUMULATE** to

NEW MICROS, INC. announces its "Generic Target Computer"T.M.
with new industrial-grade enclosure.

FORTH IN A NEMA 12 BOX



"Generic Target Computer"T.M. with NEMA 12 Case - $116.00

The NMIT-0012 "Generic Target Computer"T.M. is a digital single board computer with 5 parallel ports, 1 serial channel, 2 counter/timers, RAM, 3 JEDEC sockets, expandable, operating system and FORTH supported. Euro card format. The GTC is a minimum part configured version of the "100 Squared"T.M.

Other processors and configurations available as low as $65.

OEM Configured $230. OEM with case $255. (as shown above).

Development configured $290.

New Micros Inc.
808 Dalworth
Grand Prairie, Texas 75050
214/642-5494

```
 2  EMPTY          3  (CONVEY)       2  DOES-SIZE      79  VARIABLE       4  ("")           1  UD.            8  B/FCB         14  BLK            6  2OVER         10  ?DUP
 2  MARK           2  .TO            3  DOES-OP        69  CONSTANT      24  [COMPILE]      3  (UD.)          3  REC/BLK        8  VOC-LINK      10  2SWAP          2  FLIP
 3  HELLO          2  HOF           28  LABEL          2  RECURSIVE     27  []            13  .R             3  B/REC          8  WIDTH         62  2DUP          19  -ROT
 1  BACKGROUND:    3  CONVEY-COPY    2  UTILITY.BLK  712  :            113  '             17  .              12  B/BUF          4  'TIB          38  2DROP         41  ROT
 1  ACTIVATE       3  U/D            2  CPU8086.BLK  711  ;             12  ?MISSING       3  (.)            11  #BUFFERS      26  CONTEXT       15  2!            24  NIP
 3  SET-TASK       5  HOPPED         2  EXTEND86.BLK  65  ]              2  CRASH          3  U.R            5  QUERY          9  #VOCS         18  2@             7  TUCK
 2  TASK:          1  VIEW           2  KERNEL86.BLK  66  [              5  CONTROL        7  U.             4  TIB           11  CURRENT        2  WITHIN       100  OVER
 1  RESUME         3  @VIEW          6  VIEWS         41  DOES>         35  ASCII          3  (U.)           3  EXPECT         3  CSP            5  BETWEEN      153  SWAP
 3  DEBUG          2  COPY           5  VIEW-FILES     1  :CODE          4  DLITERAL       2  OCTAL          2  CC-FORTH      10  LAST           2  MAX          242  DUP
 1  LISTING        5  (COPY)         2  SAVE-SYSTEM    5  (;CODE)       20  LITERAL       14  DECIMAL        3  CC             5  R#             8  MIN          109  DROP
 5  SHOW           2  ESTABLISH      6  FROM           8  ;USES         48  IMMEDIATE     13  HEX            3  DEL-IN         6  DPL           11  >              2  RP!
 2  (SEMIT)        3  L              3  OPEN          50  ASSEMBLER     22  COMPILE        5  #S            31  CHAR           6  WARNING       11  <              1  RP@
 2  (PAGE)         2  B              1  DEFINE         4  (;USES)        7  EVEN          68  #              2  (CHAR)        10  STATE          5  U>             2  SP!
 2  FORM-FEED     10  N              1  B:             6  REVEAL         8  ALIGN          3  SIGN           2  CR-IN          4  PRIOR         13  U<             7  SP@
 5  PAGE           1  ::             1  A:             5  HIDE          57  C,             8  #>             2  P-IN          14  SCR            2  ?NEGATE        7  CMOVE>
 4  #PAGE          1  MANY           1  DRIVE?         5  ?CSP         103  ,              8  <#             2  RES-IN       28  EMIT           7  <>            26  CMOVE
 3  LOGO           1  TIMES          1  DIR            6  !CSP          28  ALLOT          7  HOLD           3  BACK-UP        9  PRINTING      47  =             18  C!
 2  L/PAGE         5  #TIMES         1  CREATE-FILE   53  CREATE         5  INTERPRET     12  NUMBER         2  (DEL-IN)      11  IN-FILE       17  0<>           58  C@
 4  FOOTING        3  WORDS          2  MORE           2  "CREATE        4  STATUS         2  (NUMBER)       2  BS-IN         21  FILE           5  0>           145  !
 5  INIT-PR        5  LARGEST        9  ROOT           2  .VIEW          3  ?STACK         3  NUMBER?        1  BEEP          15  HLD           14  0<           301  @
 4  EPSON          1  IND            3  -->           22  WHILE          8  DEFINED        2  (NUMBER?)      5  BACKSPACE     14  BASE          40  0=             3  (?LEAVE)
 2  SEE            3  INDEX         16  +THRU        100  ELSE           4  ?UPPERCASE     2  CONVERT       22  SPACES         5  OFFSET         7  UM/MOD         3  (LEAVE)
 5  (SEE)          3  .LINE0         5  THRU         225  IF             3  FIND           4  DOUBLE?       33  SPACE          8  #LINE          1  U*D            2  J
 3  ASSOCIATIVE:   2  TRIAD          7  ?ENOUGH       22  REPEAT        14  #THREADS       4  DIGIT         28  TYPE          15  #OUT           7  UM*           50  I
 3  CASE:          4  LIST           9  ?              8  AGAIN          3  (FIND)        25  LOAD           3  CRLF           9  DP            21  2-            14  PAUSE
 2  MAP            3  .SCR          726  (S            20  UNTIL          8  HASH           3  (LOAD)         3  (EMIT)         6  RP0           25  1-           241  NOOP
 6  OUT            4  ?CR           15  \              5  +LOOP          1  VIEW>          2  DEFAULT        3  (PRINT)        9  SP0           45  2+             1  GO
 5  DL             3  ?LINE          4  L/SCR         67  LOOP           3  >VIEW          3  VIEW#          2  PR-STAT        6  LINK          61  1+             7  PERFORM
 1  DU             3  RMARGIN       37  C/L           26  ?DO            2  >LINK          7  FLUSH         86  CR             9  ENTRY          5  8*             7  EXECUTE
 5  DUMP           4  LMARGIN        1  RECURSE       46  DO            17  >NAME          4  SAVE-BUFFERS   8  KEY            4  TOS            1  U2/           14  >NEXT
 2  .HEAD          5  HIDDEN         0  B            223  THEN          30  >BODY          3  EMPTY-BUFFERS 13  KEY?           1  */             3  2/            12  BOUNDS
 2  ?.A            1  0<=            2  Q             40  BEGIN          2  LINK>          3  IN-BLOCK       4  (CONSOLE)      2  */MOD         37  2*             4  (?DO)
 3  ?.N            3  0>=           0  DUMP           8  ?LEAVE         8  NAME>         12  BLOCK          3  (KEY)          5  MOD           39  3              4  (DO)
 2  DLN            4  >=            19  .ID           7  LEAVE          7  BODY>          3  (BLOCK)        5  (KEY?)         6  /             58  2              4  (+LOOP)
 2  EMIT.          2  <=             4  .S            14  ?<RESOLVE      6  L>NAME         3  BUFFER        26  BDOS          10  /MOD         103  1              4  (LOOP)
 3  D.2            2  U>=            4  DEPTH         8  ?<MARK         5  N>LINK         3  (BUFFER)       2  COMPARE       26  *             241  0              6  ?BRANCH
 2  .2             4  U<=            2  BYE           14  ?>RESOLVE      1  FORTH-83       2  MISSING        2  CAPS-COMP      2  MU/MOD        30  +!             6  BRANCH
 6  A              1  MS             3  START         14  ?>MARK         3  DONE?          1  DISCARD        3  COMP           3  M/MOD          6  ABS            6  (LIT)
 8  SHADOW         3  FUDGE          2  OK             2  <RESOLVE       3  TRAVERSE       4  UPDATE         7  -TRAILING      2  *D            70  -             17  UP
 3  (WHERE)        2  P!             4  INITIAL        2  <MARK         11  \S             2  ABSENT?       17  PAD            1  DMAX           8  NEGATE         6  UNNEST
 1  FIX            8  PC!            3  COLD           2  >RESOLVE     250  (              2  LATEST?       73  HERE           1  DMIN         140  +             15  EXIT
 3  EDIT           2  P@             3  WARM           2  >MARK         29  .(             4  CAPACITY       3  UPPER          2  D>            54  OFF           80  FORTH  ok
 2  ED             2  PC@            5  BOOT           9  ?CONDITION     6  >TYPE         32  DOS            2  UPC            3  D<            27  ON
 4  DONE           1  MULTI          8  QUIT           4  ABORT         11  WORD          13  SWITCH         3  MOVE           4  DU<            1  CTOGGLE
15  EDITOR         1  SINGLE         6  RUN           25  ABORT"         6  'WORD          4  FILE?          1  LENGTH         4  D=             1  CRESET
 5  DARK           1  STOP          65  IS             4  (ABORT")       6  PARSE          4  .FILE         28  COUNT          6  D0=            7  CSET
10  AT '           3  WAKE           3  (IS)           2  (?ERROR)       2  PARSE-WORD     5  WRITE-BLOCK   10  BLANK          2  ?DNEGATE      18  FALSE
 4  -LINE          3  SLEEP          6  >IS            3  ?ERROR         4  SOURCE         4  READ-BLOCK    14  ERASE          5  D-            30  TRUE
 5  BLOT           4  !LINK         16  USER           5  WHERE          2  (SOURCE)       5  >UPDATE        4  FILL           1  D2/           38  NOT
 2  REPLACE        3  @LINK          6  #USER          2  FORGET         9  PLACE         14  BUFFER#        8  CAPS           1  D2*           11  XOR
 5  INSERT         8  LOCAL        116  CODE           3  (FORGET)       5  /STRING        5  >END           4  BELL           3  DABS          51  OR
 8  DELETE         3  INT#           3  AVOC           2  TRIM           4  SCAN           9  >BUFFERS       5  BS             2  S>D           60  AND
 7  SEARCH         2  RESTART        4  2VARIABLE      4  FENCE          2  SKIP           3  INIT-R0       22  BL             5  DNEGATE        3  ROLL
 2  SCAN-1ST       2  (PAUSE)        2  2CONSTANT     61  "              1  D.R            8  FIRST          5  END?           5  D+             8  PICK
 4  FOUND          1  UNBUG         91  DEFINITIONS   67  ."             1  D.             4  >SIZE          4  #TIB           1  2ROT          21  R@
15  TO             5  BUG           21  VOCABULARY     4  ,"             3  (D.)           4  LIMIT          6  SPAN           3  4DUP          50  >R
 4  CONVEY         3  DOES?         35  DEFER          4  (.")           1  UD.R          10  DISK-ERROR    28  >IN            4  3DUP          59  R>
```

Figure Two

analyze the contents of a screen. After **LOAD** is vectored to **[LOAD]**, **LOAD**ing a screen will accumulate counts to words which appear in this screen. **THRU** can be used to analyze a range of screens in a file. Running a large number of screens through, we can get fairly representative statistics for all the commonly used Forth words.

F83 is very large, consisting of many system and application programs. It serves very well as a data base for the purposes of statistical analysis. Using the above technique, I ran all the source screens through this word processor, including seven F83 source files with 230 screens of code. There are 555 words in its **FORTH** vocabulary, and total occurrences of these words is 10,603. The result is tabulated in figure two. The words which occur most often — those counted thirty or more times — are listed in figure three.

The most obvious utility of the above analysis is that if we can arrange the dictionary so that the most frequently used words are at the very top, the speed interpretation and compilation will be significantly improved, because the dictionary searching time would be reduced. Another interesting observation is that comments were heavily used in the F83 system, using **(S**, **** and **(**. This is, of course, dictated by good programming style and in-line documentation.

(S	726	CR	86	OR	51
;	712	FORTH	80	ASSEMBLER	50
:	711	VARIABLE	79	>R	50
\	475	HERE	73	I	50
@	301	-	70	IMMEDIATE	48
(	250	CONSTANT	69	=	47
DUP	243	#	68	DO	46
0	241	LOOP	67	2+	45
NOOP	241	."	67	ROT	41
IF	225	[	66	0=	40
THEN	223	]	65	BEGIN	40
SWAP	153	IS	65	3	39
!	145	2DUP	62	NOT	38
+	140	1+	61	2DROP	38
CODE	116	"	61	2*	37
?MISSING	113	AND	60	DEFER	35
DROP	109	R>	59	ASCII	35
1	103	C@	58	SPACE	33
,	103	2	58	DOS	32
OVER	100	C,	57	CHAR	31
ELSE	100	OFF	54	>BODY	30
DEFINITIONS	91	CREATE	53	+!	30
				TRUE	30

Figure Three

```
                ok
18 LIST 19 LIST
Scr # 18          B:META86.BLK
  0 \ Statistical analysis of words                          25JAN85CHT
  1 DEFER WORK (S link --- , to do misc. works on vocabualry words)
  2 ' NOOP IS WORK
  3 : WORKS    (S -- , scan vocabulary and WORK on each word)
  4    CONTEXT @ HERE #THREADS 2* CMOVE
  5    BEGIN    HERE #THREADS LARGEST    DUP
  6    WHILE    DUP WORK    @ SWAP !    REPEAT    2DROP    ;
  7 : INIT-VIEW    (S link --- , clear a word counter.)
  8    2- OFF    ;
  9 : PRINT-VIEW    (S link --- , print ontents of a word counter.)
 10    CR    DUP 2- @ 6 .R    3 SPACES    L>NAME .ID    ;
 11 EXIT
 12 ' INIT-VIEW IS WORK    WORKS    ( Initialize all word counters)
 13 ' PRINT-VIEW IS WORK   WORKS    ( Print all word counters)
 14
 15

Scr # 19          B:META86.BLK
  0 \ New load for statistics                                23JAN85CHT
  1 : ACCUMULATE   (S --- , text interpreter to increment  counters)
  2    BEGIN    DEFINED    IF >VIEW 1 SWAP +!    ELSE DROP THEN
  3    FALSE DONE? UNTIL    ;
  4 : [LOAD]      (S n -- , interpret block n, like LOAD )
  5    FILE @ >R    BLK @ >R    >IN @ >R
  6    64 >IN ! ( Skip 0th line to avoid wrap-around.)
  7    BLK ! IN-FILE @ FILE ! ACCUMULATE R> >IN ! R> BLK !
  8    R> !FILES    ;
  9 EXIT
 10 ' [LOAD] IS LOAD   ( Use [LOAD] to do the WORKS)
 11 CAPS OFF           ( Do case sensitive compare and counting.)
 12 OPEN <file>        ( Select a source file to analyze.)
 13 1 10 THRU          ( Accumulate word statistics.)
 14  ...               ( Repeat for all source files.)
 15 ' PRINT-VIEW IS WORK    WORKS    ( Print results.)
 ok
```

Synonyms and Macros, Part 3

Benchmark Readability

Victor H. Yngve
Chicago, Illinois

One of the most frequently used timing benchmarks is the Eratosthenes Sieve program for calculating the prime numbers between three and 16,384. This program provides a convenient test bed for illustrating the use of the synonyms and macros presented in the previous articles in this series[1]. There are a number of special circumstances where synonyms and macros should be used instead of colon definitions. Two of these special circumstances will be illustrated in this article, which focuses on trying to improve the readability of the Sieve program.

The Sieve program operates with an array of byte flags initialized to true to represent the odd numbers in the given range. Starting with the flag for 3, a prime, the program tests each flag in succession. When it finds a flag that is true, that number is a prime. For each prime found, the program sets the flags to false for all succeeding odd multiples of that number, to indicate that they are not prime, and it then increments a counter that accumulates the total number of primes found. This total is then printed. Benchmark times are usually given for ten iterations of this program.

Screen 14 shows a Forth program for this benchmark that has been widely circulated outside of the Forth community, it having appeared at least twice in *BYTE*[2,3]. Call this version A. Please examine this program carefully. Note that even with the above explanation of what it is supposed to do, it would generally be judged as rather unreadable, if not opaque, even by someone with a knowledge of Forth. If you don't think so, try to find where to place a print word to print out the successive primes.

The reason for the obscurity of the program is not hard to find: It has been optimized for speed at the expense of readability. It transgresses the usual canons of good Forth programming style. In the interest of speed it puts everything in the one long word **DO-PRIME**, instead of using a number of shorter nested definitions carefully named to indicate their function in the program. With examples of Forth programming like this being widely circulated to the computing public, it is little wonder that Forth has sometimes been accused of being an unreadable language. In reality, when written in the recommended style that optimizes readability, it is among the most readable of languages.

A general method for writing readable Forth programs is to program the algorithm in words chosen to bring out how it works, thus words which refer to the concepts of the algorithm rather than to the details of its implementation in Forth. These application-specific words are then defined ultimately in terms of standard Forth words on earlier screens. This separates the program into an algorithmic part programmed largely in its own vocabulary, and an implementation part or preamble, which specifies how the named application-relative words are realized in terms of the Forth givens.

Please look now at screen 50, in which the benchmark is rewritten in words selected to indicate their function in executing the Sieve algorithm. This screen would undoubtedly be judged as quite readable by people curious to see what Forth code looks like. The preamble giving the Forth implementation of these words is on screen 49. Note that **PRINT-PRIME** can be changed from **DROP** to a print word for debugging purposes. This is version B.

For many programs we would be done at this point, but in this case we run into a special circumstance. There is the problem that version B will not run, as it contains errors. The definitions for **GET-FLAG**, **PRIME** and **FIRST-MULTIPLE** will not work as intended because the word **I** that they contain cannot be used to obtain the loop index when something else has been placed on the return stack, here the return from a colon definition. In fact, **FIRST-MULTIPLE** finds two returns covering up the index on the return stack.

To make the program work, these definitions must be deleted or commented out as shown on screen 54, and the words they contain returned to their original place in the loop, as shown on screen 55. This moves back into the loop some of the confusing low-level clutter that we had been trying to clear out, and consequently compromises its readability. But this version will run. Call it version C.

Or the words **GET-FLAG**, **PRIME**, **FIRST-MULTIPLE** and **CANCEL-MULTIPLES** could be coded as macros instead of colon definitions. Simply replace the colon in each by **MACRO** and the semicolon by **END-MACRO**. This is one of the reasons for using macros: to allow removing material from a loop in a way that will not add a return to the return stack. We will code these words as macros in version D.

Of course version C will also run slower because of the nesting and unnesting involved in calling the colon definitions. On a 4 MHz Z80 system it run in 106 seconds instead of the 71 seconds for version A, the original benchmark program, about 50% slower. This is a large difference for a speed benchmark. The program would have run even slower if we had been able to retain all of the colon definitions — about 60% slower than version A. This is a powerful disincentive operating against the use of good Forth programming style.

Something can be done, however, to overcome the slowness. This is the second special circumstance where synonyms and macros are useful. Suppose we replace the rest of the colon definitions on screen 49 by equivalent synonyms and macros, as shown on screens 56 and 57. This is version D. Version D runs in 71 seconds, the identical timing for version A, the original unreadable benchmark program. This is because version D compiles run-time code that is identical to that compiled by version A.

This means that a judicious use of synonyms and macros allows one to optimize simultaneously for speed and for readability — they provide both the speed of in-line code and the readability of colon definitions. This is another important use of synonyms and mac-

```
Screen # 14
  0 ( Eratosthenes sieve benchmark program   VERSION A  )
  1
  2 8190 CONSTANT SIZE
  3 CREATE FLAGS SIZE ALLOT
  4 : DO-PRIME   FLAGS SIZE 1 FILL
  5              0 SIZE 0
  6              DO   FLAGS I + C@
  7                   IF    I DUP + 3 + DUP I +
  8                         BEGIN   DUP SIZE <
  9                         WHILE   0 OVER FLAGS + C! OVER +
 10                         REPEAT  DROP DROP 1+
 11                   THEN
 12              LOOP
 13         . ." Primes " ;
 14 : 10-TIMES  10 0 DO DO-PRIME LOOP  ;
 15

Screen # 49
  0 ( Colon sieve 1    PREFACE  VERSION B   *errors*  2/23/85 vhy )
  1 8190  CONSTANT SIZE      CREATE FLAGS  SIZE ALLOT
  2 : FLAG-LIMIT  SIZE ;
  3 : FIRST-FLAG  0 ;
  4 : 0COUNT  0 ;
  5 : INC-COUNT  1+ ;
  6 : PRINT-COUNT  . ;
  7 : SET-TRUE        ( addr -- )     SIZE 1 FILL       ;
  8 : GET-FLAG        ( -- flag )     FLAGS I + C@      ; ( error )
  9 : PRIME           ( -- p )        I DUP + 3 +       ; ( error )
 10 : PRINT-PRIME  DROP ;
 11 : FIRST-MULTIPLE  ( p -- p m )    DUP I +           ; ( error )
 12 : SIZE<           ( m -- m f )    DUP SIZE <        ;
 13 : SET-FALSE       ( m -- m )      0 OVER FLAGS + C! ;
 14 : NEXT-MULTIPLE   ( p m -- p m' ) OVER +            ;
 15 : DROP-MULTIPLE  DROP ;

Screen # 50
  0 ( Colon Sieve 2    ALGORITHM  VERSION B  *errors* 2/23/85 vhy )
  1
  2 : CANCEL-MULTIPLES ( prime -- prime )
  3    FIRST-MULTIPLE                                    ( error )
  4       BEGIN SIZE< WHILE SET-FALSE NEXT-MULTIPLE REPEAT
  5    DROP-MULTIPLE  ;
  6
  7 : DO-PRIME  FLAGS SET-TRUE  0COUNT
  8    FLAG-LIMIT FIRST-FLAG
  9    DO    GET-FLAG                                    ( error )
 10      IF PRIME CANCEL-MULTIPLES PRINT-PRIME INC-COUNT THEN ( " )
 11    LOOP  PRINT-COUNT ." Primes "  ;
 12
 13 : 10-TIMES  10 0 DO DO-PRIME LOOP  ;
 14
 15

Screen # 54
  0 ( Colon sieve 1    PREFACE  VERSION C              2/23/85 vhy )
  1 8190  CONSTANT SIZE      CREATE FLAGS  SIZE ALLOT
  2    : FLAG-LIMIT  SIZE ;
  3    : FIRST-FLAG  0 ;
  4    : 0COUNT  0 ;
  5    : INC-COUNT  1+ ;
  6    : PRINT-COUNT  . ;
  7    : SET-TRUE        ( addr -- )     SIZE 1 FILL       ;
  8 \  : GET-FLAG        ( -- flag )     FLAGS I + C@      ;
  9 \  : PRIME           ( -- p )        I DUP + 3 +       ;
 10    : PRINT-PRIME  DROP ;
 11 \  : FIRST-MULTIPLE   ( p -- p m )    DUP I +           ;
 12    : SIZE<           ( m -- m f )    DUP SIZE <        ;
 13    : SET-FALSE       ( m -- m )      0 OVER FLAGS + C! ;
 14    : NEXT-MULTIPLE   ( p m -- p m' ) OVER +            ;
 15    : DROP-MULTIPLE  DROP ;
```

ros: They should be used in place of colon definitions in programs or parts of programs where speed is critical and where the extra memory requirements over colon definitions are of secondary importance. A speed benchmark program is a prime example. Other examples are the time-critical parts of programs that spend most of their time in inner loops.

How much extra memory is needed? Often it does not matter, because many Forth programs never find themselves bumping their heads against a memory limit. But sometimes it does matter. Exclusive of the 8190 bytes **ALLOT**ted for the array, the original benchmark program, version A, compiles in 162 bytes. Version B, the one with colon definitions that doesn't run, compiles in 455 bytes. The difference of 283 bytes, about 175% extra, represents the memory cost of readability for examples such as this. It amounts to an overhead for each extra colon definition of seven bytes plus the number of characters in the name, and an extra two bytes for each time the definition is used. It is the small cost that we usually gladly pay for the advantages of clarity of style, ease of programming, ease of checkout, ease of making modifications, self-documentation, and so on. Version C, which runs, compiles in 391 bytes, a difference from version A of 229 bytes, about 140% extra.

Now how much extra memory is needed for version D, the version with macros and synonyms? This version compiles in 493 bytes, giving an increase of only forty-eight bytes or 11% over version B, the equivalent version with colon definitions (that does not run). This is the approximate extra memory cost of using synonyms and macros instead of colon definitions in programs like this where no macro is used more than once and the macros are nested no more than two deep.

As can be calculated from the information given in the earlier articles, the cost of a synonym is two bytes less than the cost of a colon definition with one word in it, and the cost of a macro definition in an eight-bit system is one byte less than a colon definition, but each use of a macro costs an additional amount equal to two bytes less than the number of bytes needed for compiling the words in the macro.

Thus, for compactness one would normally use synonyms in place of colon definitions with one word in them, and for other words, one would normally use colon definitions instead of macros. Macros should be used in place of colon definitions where their benefits of speed and noninterference with the return stack are particularly important.

A programming style that separates the algorithmic part from the implementation part of a program has other advantages besides readability. Note how easy it would now be to change the program, for example to try a 16K array, or a bit array, or the use of variables instead of the stack to see how much slower they would be, or to try some of the schemes that have been suggested for speeding up this benchmark.

The use of comments is often recommended to improve the readability of programs, and they can be helpful. But note that version D is quite readable without any comments except the standard stack diagrams. This shows off the self-documenting ability of Forth.

The word "readability" is actually misleading because it may lead one to think of it as an absolute attribute of a programming language. In reality it is a relation between a particular program and a particular reader, and it depends on both. For example, the definitions in the implementation preamble of version D may be eminently readable by Forth programmers who understand how the algorithm is being implemented, but they are probably not as readable by members of the wider computing public who know little about Forth. The algorithm part, however, is readable to the larger audience because it does not require as much knowledge of Forth, but only of the Sieve algorithm. And it is more readable to Forth programmers as well, because its understanding requires primarily a knowledge of the Sieve algorithm, and not of the particulars of how it happens to be mapped onto Forth.

Actually there is a moveable line between the algorithm and the implementation part of a program, and it can be moved to accommodate the intended audience for the program. Our published examples of Forth code should be optimized for readability by a wider audience. This means that we should be more strict in removing the details of the Forth implementation from the algorithm section than in a program written for our own consumption. As in prose writing, one should write for one's audience. For example, **FLAG-LIMIT FIRST-FLAG DO** adds to the readability of the above program for the general computing public, but for Forth programmers familiar with arrays and the order of loop arguments in Forth, **SIZE 0 DO** might be quite acceptable instead.

Perhaps someone would like to challenge other languages to a comparison of readability benchmarks. Forth would come out quite well. The problem is, of course, that subjective judgments are involved, and such judgments depend strongly on what other languages the reader is already familiar with. But any argument that Forth has an unfair advantage because synonyms and macros are not part of the standard language, would be overlooking an important feature of Forth, its extensibility. And any argument that **SYNONYM, MACRO** and **END-MACRO** really constitute new system words like colon and semi-colon, and therefore are not fair, overlooks another important feature of Forth — the lack of a strong or impenetrable wall between the system program and the user program. **SYNONYM** is programmed in standard Forth-83; and the macro facility, though implementation specific, is the sort of thing that any average Forth programmer can easily add to a system. Unlike most other programming languages, Forth is a dynamically evolving language like English. And unlike some other programming languages, Forth does not erect barriers nor provide a single prescriptive way of programming that would limit freedom and discourage creativity.

```
Screen # 55
  0 ( Colon sieve 2    ALGORITHM  VERSION C            2/23/85 vhy )
  1
  2 : CANCEL-MULTIPLES ( prime -- prime )
  3  ( DUP I + )
  4       BEGIN SIZE< WHILE SET-FALSE NEXT-MULTIPLE REPEAT
  5    DROP-MULTIPLE  ;
  6
  7 : DO-PRIME  FLAGS  SET-TRUE  0COUNT
  8    FLAG-LIMIT FIRST-FLAG
  9    DO     FLAGS I + C@
 10     IF I DUP + 3 +
 11          DUP I + CANCEL-MULTIPLES PRINT-PRIME INC-COUNT THEN
 12    LOOP  PRINT-COUNT ." Primes "  ;
 13
 14 : 10-TIMES  10 0 DO DO-PRIME LOOP  ;
 15

Screen # 56
  0 ( Macro Sieve 1    PREFACE  VERSION D             2/23/85 vhy )
  1 8190  CONSTANT SIZE      CREATE FLAGS  SIZE ALLOT
  2 SYNONYM FLAG-LIMIT SIZE
  3 SYNONYM FIRST-FLAG 0
  4  SYNONYM 0COUNT 0
  5  SYNONYM INC-COUNT 1+
  6  SYNONYM PRINT-COUNT .
  7 MACRO SET-TRUE       ( addr -- )    SIZE 1 FILL        END-MACRO
  8 MACRO GET-FLAG       ( -- flag )    FLAGS I + C@       END-MACRO
  9 MACRO PRIME          ( -- p )       I DUP + 3 +        END-MACRO
 10  SYNONYM PRINT-PRIME DROP
 11 MACRO FIRST-MULTIPLE ( p -- p m )   DUP I +            END-MACRO
 12 MACRO SIZE<          ( m -- m f )   DUP SIZE <         END-MACRO
 13 MACRO SET-FALSE      ( m -- m )     0 OVER FLAGS + C!  END-MACRO
 14 MACRO NEXT-MULTIPLE ( p m -- p m' ) OVER +             END-MACRO
 15  SYNONYM DROP-MULTIPLE DROP

Screen # 57
  0 ( Macro Sieve 2    ALGORITHM  VERSION D            2/23/85 vhy )
  1
  2 MACRO CANCEL-MULTIPLES ( prime -- prime )
  3    FIRST-MULTIPLE
  4       BEGIN SIZE< WHILE SET-FALSE NEXT-MULTIPLE REPEAT
  5    DROP-MULTIPLE  END-MACRO
  6
  7 : DO-PRIME  FLAGS SET-TRUE  0COUNT
  8    FLAG-LIMIT FIRST-FLAG
  9    DO     GET-FLAG
 10       IF PRIME CANCEL-MULTIPLES PRINT-PRIME INC-COUNT THEN
 11    LOOP  PRINT-COUNT ." Primes "  ;
 12
 13 : 10-TIMES  10 0 DO DO-PRIME LOOP  ;
 14
 15
```

References

1. Yngve, Victor H. "Synonyms and Macros," parts 1 and 2. *Forth Dimensions* VII/3 (September/October 1985).

2. Gilbreath, Jim. "A High-Level Language Benchmark." *BYTE* Vol. 6, No. 9, p. 180 (September 1981).

3. Tello, Ernie. "polyFORTH and PC/FORTH." *BYTE* Vol. 9, No. 12, p. 303 (November 1984).

The ForthCard™

STAND ALONE OPERATION

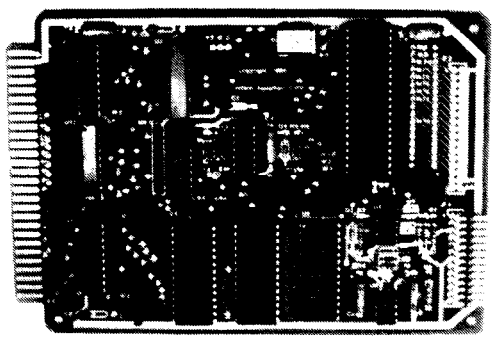
STD BUS INTERFACE

EPROM/EEPROM PROGRAMMER

RS-232 I/O

PARALLEL I/O

ROCKWELL FORTH CHIP



Evaluation Unit **$299**
Part #STD65F11-05 includes:
ForthCard, Development ROM, 8Kbyte RAM, Manuals

OEM Version as low as **$199**
Part #STD65F11-00 does not include memory or manuals

The Forthcard provides OEMs and end users with the ability to develop Forth and assembly language programs on a single **STD bus compatible** card.

Just add a CRT terminal (or a computer with RS-232 port), connect 5 volts and you have a **self contained Forth computer.** The STD bus interface makes it easy to expand.

Download Forth source code using the serial port on your PC. Use the **onboard EPROM/EEPROM programming** capability to save debugged Forth and assembly language programs. Standard UV erasable EPROMs may also be programmed with an external Vpp supply.

NEW! Options and Application Notes

Electrically Eraseable PROMs (EEPROMs)

FREEZE the dictionary in EEPROM (save in non-volatile memory, to be restored on power up)

Download Software for your IBM PC or CP/M

Non-Volatile CMOS RAM with battery 2K, 8K, optional Clock/calendar

Fast 2MHz clock (4MHz crystal)

Disk Controller Card (5¼")

Self Test Diagnostics

Parallel printer interface

Ask about our ForthBox™
A complete STD bus oriented system including the ForthCard, Disk Controller, Disk Drive(s), STD Card Cage, Cabinet and power supply.

CALL TODAY FOR COMPLETE INFORMATION!

HiTech Equipment Corporation

9560 Black Mountain Road
San Diego, CA 92126
(619) 566-1892

MasterCard VISA

FORTH INTEREST GROUP MAIL ORDER FORM

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

MEMBERSHIP

IN THE FORTH INTEREST GROUP

107 - MEMBERSHIP in the FORTH INTEREST GROUP & Volume 7 of FORTH DIMENSIONS. No sales tax, handling fee or discount on membership. See the back page of this order form.

The Forth Interest Group is a worldwide non-profit member-supported organization with over 5,000 members and 80 chapters. FIG membership includes a subscription to the bi-monthly publication, FORTH Dimensions. FIG also offers its members publication discounts, group health and life insurance, an on-line data base, a large selection of Forth literature, and many other services. Cost is $20.00 per year for USA, Canada & Mexico; all other countries may select surface ($27.00) or air ($33.00) delivery.

The annual membership dues are based on the membership year, which runs from May 1 to April 30.

When you join, you will receive issues that have already been circulated for the current volume of Forth Dimensions and subsequent issues will be mailed to you as they are published.

You will also receive a membership card and number which entitles you to a 10% discount on publications from FIG. Your member number will be required to receive the discount, so keep it handy.

HOW TO USE THIS FORM

1. Each item you wish to order lists three different Price categories:

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

2. Select the item and note your price in the space provided.

3. After completing your selections enter your order on the fourth page of this form.

4. Detach the form and return it with your payment to **The Forth Interest Group**.

FORTH DIMENSIONS BACK VOLUMES

The six issues of the volume year (May - April)

101 - Volume 1 FORTH Dimensions (1979/80) $15/16/18 _____

102 - Volume 2 FORTH Dimensions (1980/81) $15/16/18 _____

103 - Volume 3 FORTH Dimensions (1981/82) $15/16/18 _____

104 - Volume 4 FORTH Dimensions (1982/83) $15/16/18 _____

105 - Volume 5 FORTH Dimensions (1983/84) $15/16/18 _____

106 - Volume 6 FORTH Dimensions (1984/85) $15/16/18 _____

ASSEMBLY LANGUAGE SOURCE CODE LISTINGS

Assembly Language Source Listings of fig-Forth for specific CPUs and machines with compiler security and variable length names.

513 - 1802/MARCH 81 ..... $15/16/18 _____

514 - 6502/SEPT 80 ..... $15/16/18 _____

515 - 6800/MAY 79 ..... $15/16/18 _____

516 - 6809/JUNE 80 ..... $15/16/18 _____

517 - 8080/SEPT 79 ..... $15/16/18 _____

518 - 8086/88/MARCH 81 ..... $15/16/18 _____

519 - 9900/MARCH 81 ..... $15/16/18 _____

520 - ALPHA MICRO/SEPT 80 ..... $15/16/18 _____

521 - APPLE II/AUG 81 ..... $15/16/18 _____

522 - ECLIPSE/OCT 82 ..... $15/16/18 _____

523 - IBM-PC/MARCH 84 ..... $15/16/18 _____

524 - NOVA/MAY 81 ..... $15/16/18 _____

525 - PACE/MAY 79 ..... $15/16/18 _____

526 - PDP-11/JAN 80 ..... $15/16/18 _____

527 - VAX/OCT 82 ..... $15/16/18 _____

528 - Z80/SEPT 82 ..... $15/16/18 _____

BOOKS ABOUT FORTH

200 - ALL ABOUT FORTH ..... $25/26/35 _____
Glen B. Haydon
An annotated glossary for MVP Forth; a 79-Standard Forth.

205 - BEGINNING FORTH ..... $17/18/21 _____
Paul Chirlian
Introductory text for 79-Standard.

215 - COMPLETE FORTH ..... $16/17/20 _____
Alan Winfield
A comprehensive introduction including problems with answers. (Forth 79)

220 - FORTH ENCYCLOPEDIA ..... $25/26/35 _____
Mitch Derick & Linda Baker
A detailed look at each FIG-Forth instruction.

225 - GORTH FUNDAMENTALS, V. 1 ..... $16/17/20 _____
Kevin McCabe
A textbook approach to 79 Standard Forth.

230 - FORTH FUNDAMENTALS, V. 2 ..... $13/14/16 _____
Kevin McCabe
A glossary.

232 - FORTH NOTEBOOK ..... $25/26/35 _____
Dr. C. H. Ting
Good examples and applicatoins. Great learning aid. PolyFORTH is the dialect used. Some conversion advice is included. Code is well documented.

233 - FORTH TOOLS ..... $19/21/23 _____
Gary Feierbach & Paul Thomas
The standard tools required to create and debug Forth-based applications.

235 - INSIDE F—83 ..... $25/26/35 _____
Dr. C. H. Ting
Invaluable for those using F-83.

237 - LEARNING FORTH ..... $17/18/21 _____
Margaret A. Armstrong
Interactive text, introduction to the basic concepts of Forth. Includes section on how to teach children Forth.

240 - MASTERING FORTH ..... $18/19/22 _____
Anita Anderson & Martin Tracy (MicroMotion)
A step-by-step tutorial including each of the commands of the Forth-83 International Standard; with utilities, extensions and numerous examples.

245 - STARTING FORTH (soft cover) ..... $20/21/24 _____
Leo Brodie (FORTH, Inc.)
A lively and highly readable introduction with exercises.

246 - STARTING FORTH (hard cover) ..... $24/25/29 _____
Leo Brodie (FORTH, Inc.)

255 - THINKING FORTH (soft cover) ..... $16/17/20 _____
Leo Brodie
The sequel to "Starting Forth". An intermediate text on style and form.

265 - THREADED INTERPRETIVE LANGUAGES $23/25/28 _____
R.G. Loeliger
Step-by-step development of a non-standard Z-80 Forth.

270 - UNDERSTANDING FORTH ..... $3.50/5/6 _____
Joseph Reymann
A brief introduction to Forth and overview of its structure.

FORML CONFERENCE PROCEEDINGS

FORML PROCEEDINGS - FORML (the Forth Modification Laboratory) is an informal forum for sharing and discussing new or unproven proposals intended to benefit Forth. Proceedings are a compilation of papers and abstracts presented at the annual conference. FORML is part of the Forth Interest Group

310 - FORML PROCEEDINGS 1980 ..... $25/28/35 _____
Technical papers on the Forth language and extensions.

311 - FORML PROCEEDINGS 1981 (2V) ..... $40/43/45 _____
Nucleus layer, interactive layer, extensible layer, metacompilation, system development, file systems, other languages, other operating systems, applications and abstracts without papers.

312 - FORML PROCEEDINGS 1982 ..... $25/28/35 _____
Forth machine topics, implementation topics, vectored execution, system development, file systems and languages, applications.

313 - FORML PROCEEDINGS 1983 ..... $25/28/35 _____
Forth in hardware, Forth implementations, future strategy, programming techniques, arithmetic & floating point, file systems, coding conventions, functional programming, applications.

314 - FORML PROCEEDINGS 1984 ..... $25/28/35 _____
Expert systems in Forth, using Forth, philosophy, implementing Forth systems, new directions for Forth, interfacing Forth to operating systems, Forth systems techniques, adding local variables to Forth.

ROCHESTER PROCEEDINGS

The Institute for Applied Forth Research, Inc. is a non-profit organization which supports and promotes the application of Forth. It sponsors the annual Rochester Forth Conference.

321 - ROCHESTER 1981 (Standards Conference) $25/28/35 _____
79-Standard, implementing Forth, data structures, vocabularies, applications and working group reports.

322 - ROCHESTER 1982
(Data bases & Process Control) ..... $25/28/35 _____
Machine independence, project management, data structures, mathematics and working group reports.

323 - ROCHESTER 1983 (Forth Applications) . $25/28/35 _____
Forth in robotics, graphics, high-speed data acquisition, real-time problems, file management, Forth-like languages, new techniques for implementing Forth and working group reports.

324 - ROCHESTER 1984 (Forth Applications) . $25/28/35 _____
Forth in image analysis, operating systems, Forth chips, functional programming, real-time applications, cross-compilation, multi-tasking, new techniques and working group reports.

325 - ROCHESTER 1985
(Software Management and Engineering) $25/28/35 _____
Improving software productivity, using Forth in a space shuttle experiment, automation of an airport, development of MAGIC/L, and a Forth-based business applications language, includes working group reports.

THE JOURNAL OF FORTH APPLICATION & RESEARCH

A refereed technical journal published by the Institute for Applied Forth Research, Inc.

401 - JOURNAL OF FORTH RESEARCH V.1 #1 $15/16/18 _____
Robotics.

402 - JOURNAL OF FORTH RESEARCH V.1 #2 $15/16/18 _____
Data Structures.

403 - JOURNAL OF FORTH RESEARCH V.2 #1 $15/16/18 _____
Forth Machines.

404 - JOURNAL OF FORTH RESEARCH V.2 #2 $15/16/18 _____
Real-Time Systems.

405 - JOURNAL OF FORTH RESEARCH V.2 #3 $15/16/18 _____
Enhancing Forth.

406 - JOURNAL OF FORTH RESEARCH V.2 #4 $15/16/18 _____
Extended Addressing.

407 - JOURNAL OF FORTH RESEARCH V.3 #1 $15/16/18 _____
Forth-based laboratory systems and data structures.

REPRINTS

420 - BYTE REPRINTS ..... $5/6/7 _____
Eleven Forth articles and letters to the editor that have appeared in *Byte* magazine.

421 - POPULAR COMPUTING 9/83 ..... $5/6/7 _____
Special issue on various computer languages, with an in-depth article on Forth's history and evolution.

DR. DOBB'S

This magazine produces an annual special Forth issue which includes source-code listings for various Forth applications.

422 - DR. DOBB'S 9/82 ..... $5/6/7 _____

423 - DR. DOBB'S 9/83 ..... $5/6/7 _____

424 - DR. DOBB'S 9/84 ..... $5/6/7 _____

HISTORICAL DOCUMENTS

501 - KITT PEAK PRIMER ..... $25/27/35 _____
One of the first institutional books on Forth. Of historical interest.

502 - FIG-FORTH INSTALLATION MANUAL .. $15/16/18 _____
Glossary model editor - We recommend you purchase this manual when purchasing the source-code listings.

503 - USING FORTH ..... $20/21/23 _____
FORTH, Inc.

REFERENCE

305 - FORTH 83 STANDARD ..... $15/16/18 _____
The authoritative description of 83-Standard Forth. For reference, not instruction.

300 - FORTH 79 STANDARD ..... $15/16/18 _____
The authoritative description of 79-Standard Forth. Of historical interest.

316 - BIBLIOGRAPHY OF FORTH REFERENCES
2nd edition, Sept. 1984 ..... $15/16/18 _____
An excellent source of references to articles about Forth throughout microcomputer literature. Over 1300 references.

MISCELLANEOUS

601 - T-SHIRT SIZE _____
Small, Medium, Large and Extra-Large.
White design on a dark blue shirt. $10/11/12 _____

602 - POSTER (BYTE Cover) ..... $15/16/18 _____

616 - HANDY REFERENCE CARD ..... FREE _____

683 - FORTH-83 HANDY REFERENCE CARD ..... FREE _____

SELECTED PUBLICATIONS

The following publications are NEW additions to the Forth Interest Group Order Form as selected by the FIG Publications Committee.

FORTH NOTEBOOK by Dr. C. H. Ting.
Good examples and applications. Great learning aid. PolyFORTH is the Forth dialect used. Some conversion advice is included in this book. Code is well documented. The first part has 10 games written in FORTH. It includes some tools for control applications. There is a section on image processing and then other forms of utilities. It includes a general tutorial on Forth which may be shown on an overhead projector and used with lecture notes.

INSIDE F-83 by Dr. C. H. Ting
Invaluable for those using F-83, this is the only documentation for the F-83 system. The book includes 4 sections: 1) a tutorial to the F-83 system; 2) detailed discusion of the Forth kernel; 3) Program run utilities; and 4) 8086 specific utilities. This edition was published in June, 1985.

1984 ROCHESTER CONFERENCE PROCEEDINGS
(Software Management and Engineering)
The fifth annual Rochester Forth Conference was held June 12-15, 1985 at the University of Rochester in New York. The Proceedings appear as a special 266 page issue of Volume 3 of the Journal of Forth Application and Research. A focus of the conference was on how to improve software productivity. Invited papers discuss the use of Forth in a space shuttle experiment, using Forth in the automation of an airport, Forth in the development of MAGIC/L and a Forth-based business applicatoins language. The Proceedings incudes 50 papers and working reports.

JOURNAL OF FORTH APPLICATION AND RESEARCH V. 3, #1
This is the latest issue of the Journal and features Forth-based laboratory systems and data structures.

FORML CONFERENCE PROCEEDINGS, 1980 through 1984
Prices as shown on this order form will increase by $5.00 as of January 1, 1986. Order now and save on items 310 - 314.

PUBLICATIONS SURVEY

If you would like to suggest any other publication for review by the FIG Publications committee for inclusion in the Forth Interest Group Order Form, please complete the information below and return to FIG.

Title: _____

Author: _____ Publisher: _____

Comments: _____

_____

Your comments on any of the publications we currently carry are most welcome, please complete information below.

Title: _____

Comments: _____

_____

FORTH INTEREST GROUP

P.O. BOX 8231 SAN JOSE, CALIFORNIA 95155 408/277-0668

Name _____

Company _____

Address _____

City _____

State/Prov. _____ **ZIP** _____

Country _____

Phone _____

OFFICE USE ONLY

By _____ Date _____ Type _____

Shipped By _____ Date _____

UPS Wt. _____ Amt. _____

USPS Wt. _____ Amt. _____

BO Date _____ Wt. _____ Amt. _____

By _____

ITEM #	TITLE	AUTHOR	QTY	UNIT PRICE	TOTAL
107	MEMBERSHIP				SEE BELOW

☐ Check enclosed (payable to: **FORTH INTEREST GROUP**)

☐ VISA ☐ MASTERCARD

Card # _____

Expiration Date _____

Signature _____

SUBTOTAL	
10% MEMBER DISCOUNT MEMBER # _____	
CA. RESIDENTS SALES TAX	
HANDLING FEE	**$2.00**
MEMBERSHIP FEE $20/27/33 ☐ NEW ☐ RENEWAL	
TOTAL	

PAYMENT MUST ACCOMPANY ALL ORDERS

MAIL ORDERS
Send to:
Forth Interest Group
P.O. Box 8231
San Jose, CA 95155

PHONE ORDERS
Call 408/277-0668 to place credit card orders or for customer service. Hours: Monday-Friday, 9am-5pm PST.

PRICES
All orders must be prepaid. Prices are subject to change without notice. Credit card orders will be sent and billed at current prices. $15 minimum on charge orders. Checks must be in US$, drawn on a US Bank. A $10 charge will be added for returned checks.

POSTAGE & HANDLING
Prices include shipping. A $2.00 handling fee is required with all orders.

SHIPPING TIME
Books in stock are shipped within five days of receipt of the order. Please allow 4-6 weeks for out-of-stock books (delivery in most cases will be much sooner).

SALES TAX
California deliveries add 6%. San Francisco Bay Area add 7%.

11/85

Extending the Multi-Tasker

Mailboxes

R.W. Dobbins
Columbia, Maryland

The multi-tasker presented by H. Laxen[1] is an excellent way to harness the full power of Forth, particularly for real-time applications. It is also an adequate starting point for more comprehensive multi-tasking schemes. A need often arises in such systems for independent tasks to exchange information. In this article, some ideas are presented on how the multi-tasker can be extended to incorporate inter-task communication and cooperation.

Communication by Mailboxes

Tasks can communicate in various ways. The most obvious method is to have one or more common variables which tasks use to pass information. This would be analogous to using variables rather than the stack to pass parameters between words. Apart from negating much of the true usefulness of Forth, this approach causes other difficulties: words are difficult to test or modify, since dependencies buried in the data may be obscure. With multi-tasking, these problems become even more significant, so a more structured approach must be sought.

The method chosen here uses "mailboxes" to allow tasks to communicate in a simple and straightforward fashion. Assume that a mailbox is initially empty and that task A wants to send messages to a receiving task B. Ideally, each time A sends a message the mailbox is empty, while there is always a message in the mailbox when B attempts to fetch one. Unfortunately, things rarely work out this well in practice. Either A will produce messages faster than they can be consumed by B, or B will try to fetch messages faster than A can produce them. Let us consider how each of these problems can be dealt with.

If B finds the mailbox empty, it must somehow wait until a message arrives from A. This is where the original multi-tasker comes into play, since we can make use of the word **PAUSE** to have B temporarily relinquish control until A has deposited a message in the mailbox.

What if A finds a message already in the mailbox when wanting to send another message? There are several possible ways to deal with this problem:

- the sending task pauses until the mailbox is cleared
- the new message replaces the old one
- messages are stacked in the mailbox

Each of these strategies may be valid in a particular situation and there is no clear choice as to which one is "correct." However, we will describe the first method, since there are some important advantages, *viz.*:

- simplicity, as demonstrated by the listing
- efficiency; especially if rewritten in assembly language, the overhead involved in polling the mailbox is very small
- no elaborate data structures, just a single word in memory, so the technique can be easily applied in many different situations, even in the direct control of I/O devices

In short, the Forth approach has been adopted. Implementations of the other schemes are straightforward extensions of the basic method.

```
: MAILBOX  ( define a mailbox variable
and initialize it )
   CREATE  0  ,  DOES>  ;
: SEND     ( n MAILBOX --- )
   BEGIN  PAUSE  DUP
   AT  0=  UNTIL  !  ;
: RECEIVE     ( MAILBOX --- n )
   BEGIN  PAUSE  DUP  AT
   ?DUP  UNTIL  0  ROT  !  ;
```

MAILBOX is really just a variable which holds the message, initially zero. **SEND** waits until the mailbox has been cleared, then deposits the new message. **RECEIVE** examines the mailbox and waits until a (non-zero) message has arrived before returning with the message on the stack. Note that a message is "removed" from the mailbox, not simply read, so it is not possible for a task to mistakenly read the same message twice.

Now, to see these words in action consider the following simple but representative problem. You are to sample an eight-bit analog-to-digital converter (ADC) ten times per second and collect approximately 10,000 readings on disk. Assume that the word **READ** has already been defined to fetch a value from the ADC after a 100 millisecond delay. First of all, the following single-task solution will *not* work correctly, which is why we are considering multi-tasking in the first place.

```
1  CONSTANT  DATA  ( disk block to
store data )
10  CONSTANT  LAST  ( last disk
block )
:  SAMPLER
   LAST  1+  DATA  DO  I  BLOCK
   1024  OVER  +  SWAP
   DO  READ  I  C!
   LOOP  UPDATE  LOOP  ;
```

The difficulty with this solution is that at the end of the loop, data has to be written to disk. Quite likely, this will take longer than the required 100 millisecond sampling period. In fact, worse still is that since **READ** has a built-in delay, this solution only works if the disk access requires approximately zero time! So it is apparent that some data will be lost with this method and another approach must be found.

In the multi-tasking solution, one would have a task to sample the ADC, while a second task would handle the formatting and writing of the data to disk. This approach does not suffer from the above problem because while the **WRITER** is busy updating the previous disk block, **SAMPLER** is gathering the next sample in parallel:

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.

```
MAILBOX  SAMPLE
400  TASK:  SAMPLING
400  TASK:  WRITING
:  SAMPLER
   SAMPLING  ACTIVATE
   BEGIN  READ  SAMPLE  SEND
   AGAIN  ;
:  WRITER
   WRITING  ACTIVATE
   LAST  1+  DATA
   DO  I  BLOCK
   1024  OVER  +  SWAP
   DO  SAMPLE  RECEIVE
   I  C!  LOOP  UPDATE
   LOOP  STOP  ;
```

Another benefit of multi-tasking lies in the fact that each task can be run separately in typical Forth modular style, and can thus be tested more easily. We can run the **SAMPLER** to make sure that it reads samples correctly, by examining the value in **SAMPLE** from the keyboard. Next, **WRITER** could be run and tested on its own. Dummy data could be supplied from the keyboard and the correct disk accessing could be observed and checked. Finally, we can connect the two working tasks together via the mailbox and we have a working system. Contrast this with the original version of **SAMPLER**, which would have been more difficult to verify. Bear in mind that this is a rather trivial example. In a more realistic application, the benefits of multi-tasking in the testing and debugging process would become even more marked.

References

1. Laxen, H. "Multi-tasking," *Forth Dimensions* V/4,5 (November/December 1983 and January/February 1984).

2. Brinch-Hansen, P. *Operating System Principles*, Prentice-Hall, 1973.

Atari Painting Forth

Stephen James
Ventura, California

Paint with the power of Forth. Splash vivid hues with your Atari. Create alien worlds and magical kingdoms — fast and colorful. With the Forth source code in this article, you — the computer artist — will draw and paint beautiful graphics in Atari modes 3, 5 and 7. Soon you'll find yourself composing multicolored displays for an adventure game, slide show or arcade screen. The Forth code includes various pens and brushes for designing complex graphic art. Pens sketch fine lines and brushes add color texture, in an infinite variety of color combinations.

Where's the Tip?

Instead of using a camel-hair brush to compose dungeon scenes or space art, you use the computer's cursor and a joystick plugged into port one. An artist sets up his drawing board and places his pens and brushes in a favorite spot. Similarly, as you turn on your computer easel, the cursor lies in the upper left-hand corner of the monitor, coordinates 0,0. If this location is not handy, change the coordinates in the code.

After setup, the artist might sketch or paint a mountain peak in the background and a wind-swept lake in the foreground. But always, at some point, the artist lifts the tip of the pen or brush from one spot and moves it over the art to another spot. If he didn't lift the tip, the picture would be ruined. Likewise, your computer tip, displayed as a cursor, must sometimes be seen but not felt. In other words, you must be able to see the tip and move it on the monitor, but it must not leave any marks.

This is accomplished by plotting the tip's position with a non-background color to visibly locate the tip's position. Before the tip moves to its new position, its old position is erased: the original color is restored.

This type of plotting/erasing works as long as the plotted color is not the same as the position's original color. But this is not always the case. When the plotted color duplicates the color of an underlying forest, stream or building, the tip's position hides.

A flashing technique eliminates this problem. Plotting the tip's position with one color, then another, and so on, produces multiple blinking colors. Therefore, it doesn't matter what the original color is, because at some time during the flashing cycle the cursor's color does not match the underlying color or texture.

With the tip's position being completely defined, now you can answer, "Where's the tip?" Type in the Forth word **TIP**. Use the joystick to move the tip across the monitor without leaving a trace. Then, like an artist, find a spot to draw that lake, and set the tip down by pressing the joystick's button.

A Fine Line

Although it belongs with your art supplies, **TIP** does not draw. But its code serves as a seedling for the Forth word **PEN**, which lets you draw with a fine line.

You draw by plotting the tip position, as previously explained, but without erasing any position to which it moves. That way, the plotted color remains. By using the joystick, you sketch various scenes with vivid color. However, as your artistic talent comes to life, there will be an abundance of color on the screen, making the pen's position invisible.

If you stop drawing for a moment, it is hard to determine where the pen is. Though you see the moving pen, the pen hides when at rest — especially if it intersects a line of the same color. Therefore, even **PEN** uses the flashing technique to show the cursor's position.

In graphic modes 3, 5 and 7, you draw with four colors stored in the computer's color registers. These are accessed by **PEN**. Precede **PEN** with register number 0, 1, 2 or 3, remembering that register zero controls background color.

With **PEN**, you soon find yourself drawing monsters and space stations. Like **TIP**, **PEN** disengages when you press the joystick trigger.

A Wider Spread

As a computer artist, you fill in oceans with shades of blue, grass with shades of green and hills with shades of brown. But filling shapes with a fine line is too slow. A wider spread is needed.

With Forth, you simply trade **PEN** for **BRUSH**, and splash paint on the electronic canvas, using wide strokes. By expanding **PEN** into **BRUSH**, scenes fill in fast. Four pixels (the tiny points of light on a tv screen) are plotted, instead of only one. Thus, **BRUSH** leaves a trail the size of four pixels: two pixels high and two pixels wide, twice the width and height of the **PEN** line. Note that pixel size varies with each graphic mode, so fastest filling is done in mode 3.

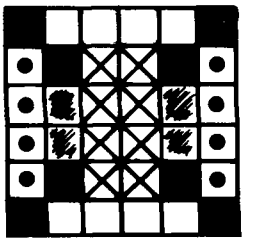
Holding your brush with the keyboard, you now fill your palette with various textures or color patterns. You create each texture by placing different plot structures within a matrix of pixels, the patterns ranging from subtle to vibrant. The images in Figure One show sample combinations, although an infinite variety is available.

Along with differing texture structures, numerous combinations of the four available hues add variety:

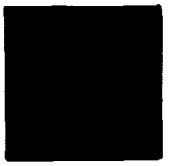
Combination	Color Values
1	0 and 1
2	0 and 2
3	0 and 3
4	1 and 2
5	1 and 3
6	2 and 3

Thus, if green is stored in color value 1 and brown is stored in 2, a color mix of green/brown squeezes out of a Forth paint tube.

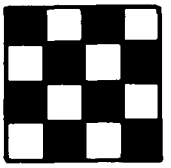
SAMPLE



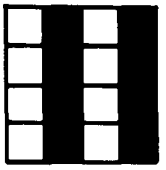
THE ERASER



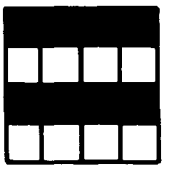
SOLID



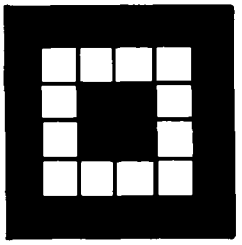
DIAGONAL (XBRUSH)



VERTICAL (VBRUSH)



HORIZONTAL (HBRUSH)



BLOCK

To experiment, type **1 2 XBRUSH** while in graphic mode 3, and examine the texture. Next, press the joystick trigger and examine the texture in other modes. Then change the color registers preceding **XBRUSH**. After you have seen various color combinations, replace the Forth word **XBRUSH** with **VBRUSH** or **HBRUSH** to see other textures.

Solid color spreads are also possible. They are a subset of textures, since painting with one hue requires a texture Forth word. However, the color combination consists of duplicate colors:

Combination	Color Values
7	0 and 0
8	1 and 1
9	2 and 2
10	3 and 3

For example, typing **2 2 XBRUSH** spreads a solid color consisting of the hue stored in color register 2, whereas typing **3 3 XBRUSH** produces pure register 3.

Besides solids and two-color textures, combining three or four colors with unique pixel matrices gives your art an added touch. There are endless possibilities, only one of which is shown in Figure Two.

The Eraser

Unlike a water colorist, you will erase not only pen marks, but brush strokes, too. This is an added benefit of computer painting. Just type **0 PEN** and use the joystick to erase details; otherwise, use **0 0 XBRUSH** to erase large sections.

Paint Source Code

With the Forth Paint source code listings, you have the foundation for a digitized Rembrandt. Venture beyond the limits.

Screen 10 defines various joystick constants. Instead of using actual numeric values, constants are used to increase speed. Lines 2 - 12 correspond to the direction values seen in joystick hardware register 632. Push the joystick up while fetching the value from 632, and the result will be 14.

Screen 11 defines words that will automatically fetch values from the computer. Even-numbered lines from 2 - 8 fetch values related to a joystick's column — what direction it leans. Odd-numbered lines from 3 - 9 fetch numbers that determine whether or not the respective joystick triggers are pressed.

Screen 12 sets a time delay. **MSEC** equates to a fraction of a second. Line 3 enables the time delay to be a function of a user-supplied value placed on the stack. Execute **25 MSEC** and then **250 MSEC**, and you should be able to notice the difference in time delay.

Screens 13 and 14 provide input for updating the x,y coordinates of a joystick. When this data is any number other than zero, the cursor — tied to the joystick — moves. Line 3 places a

```
Scr # 10
  0  ( stick constants                )
  1
  2 15 CONSTANT STOP
  3 14 CONSTANT FD    ( forward      )
  4 14 CONSTANT UP
  5 13 CONSTANT BD    ( backward     )
  6 13 CONSTANT DN
  7  7 CONSTANT RT    ( right        )
  8 11 CONSTANT LT    ( left         )
  9  6 CONSTANT FDRT  ( fwd-rght     )
 10  5 CONSTANT BDRT  ( bck-rght     )
 11 10 CONSTANT FDLT  ( fwd-left     )
 12  9 CONSTANT BDLT  ( bck-left     )
 13
 14
 15                                 -->

Scr # 11
  0  ( joystick utilities            )
  1
  2 : 0STICK    632 C@ ;
  3 : 0STRIG    644 C@ ;
  4 : 1STICK    633 C@ ;
  5 : 1STRIG    645 C@ ;
  6 : 2STICK    634 C@ ;
  7 : 2STRIG    646 C@ ;
  8 : 3STICK    635 C@ ;
  9 : 3STRIG    647 C@ ;
 10
 11
 12
 13
 14
 15
```

fetched joystick value on the Forth return stack. Line 4 checks for a joystick that is in the straight-up position. This indicates to the computer that the operator wants the cursor movement to halt. Thus, when this condition is true, the next input for processing is 0,0. The cursor moves neither left, right, up nor down. Line 5 checks for a forward joystick movement. If valid, the y coordinate changes so that the cursor moves straight up the screen.

Continuing on screen 14, lines 2 - 8 are like the conditional statements of screen 13, except diagonal directions are checked. Lines 9 - 10 end the **IF ELSE** statements. Finally, the original joystick value is pulled off the return stack and dropped.

Screen 40 defines the paint variables. In lines 2 - 3, **STKX** and **STKY** correspond to the x,y coordinates of the joystick's position. They are set to x=0 and y=0 when the screen is first loaded. Lines 5 - 6 define **CLRA** (color A) and **CLRB** (color B). **CLRA** is the variable which stores the value of the chosen pen color. **CLRB** is a variable which stores the second color when multi-colored brush textures are used. Line 8 defines the variable **LOCCLR** (color location). It stores the value of the color pixel that the tip is over. Then, when the flashing tip moves to another pixel, the original pixel color returns. Line 10 defines the word that will place the cursor in the upper-left corner of the screen. It is a good idea to

```
Scr # 12
  0
  1
  2 : MSEC
  3 (   ) 0 DO
  4 LOOP ;
  5
  6
  7
  8
  9
 10
 11
 12
 13
 14
 15

Scr # 13
  0  ( fetch joystick coordinates )
  1
  2 : OSTKXY   (        --- x y )
  3 OSTICK >R
  4 R STOP = IF   0   0
  5 ELSE
  6 R FD   = IF   0  -1
  7 ELSE
  8 R RT   = IF   1   0
  9 ELSE
 10 R BD   = IF   0   1
 11 ELSE
 12 R LT   = IF  -1   0
 13 ELSE
 14
 15                              ==>
```

use **ZERO** after initializing any graphic mode.

On screen 41, **FLASHSTK** flashes the tip's position on the monitor. Line 2 gathers the color that the tip is over by using the BASIC-like command **LOCATE**. This color is stored in the variable **LOCCLR**. Lines 3 - 9 plot a different color after a twenty-five millisecond delay (see **MSEC** defined in screen 12). This delay is necessary, otherwise the color change would be too fast for the eye. Lines 10 - 12 ensure that before **FLASHSTK** ends, the original color stored in **LOCCLR** is returned (plotted). That way, when you exit from any paint routine, the original color is not altered.

Screen 42 defines **TIP** (tip position), which moves the tip on the monitor, corresponding to the joystick. Line 3 begins a loop which will continue **UNTIL** (line 8) the trigger is pressed (line 7). **FLASHSTK** flashes the tip's position, as already defined. Line 5 fetches the last position of the tip. **OSTK** leaves the latest joystick values on the stack. Line 6 rotates the stack values for ease of manipulation and stores the new joystick coordinates in **STKX** and **STKY**.

```
Scr # 14
  0  ( continue                          )
  1
  2 R FDRT = IF   1  -1
  3 ELSE
  4 R BDRT = IF   1   1
  5 ELSE
  6 R BDLT = IF  -1   1
  7 ELSE
  8 R FDLT = IF  -1  -1
  9 THEN THEN THEN THEN THEN
 10 THEN THEN THEN THEN  R> DROP ;
 11
 12
 13
 14
 15

Scr # 40
  0  ( Paint variables                   )
  1
  2 0 VARIABLE STKX
  3 0 VARIABLE STKY
  4
  5 0 VARIABLE CLRA
  6 0 VARIABLE CLRB
  7
  8 0 VARIABLE LOCCLR
  9
 10 : ZERO
 11  0 STKX !
 12  0 STKY ! ;
 13
 14
 15                                   -->
```

Screen 43 defines **PEN**. Graphic modes 3, 5 and 7 give four colors from which to choose. This allows four color pens for drawing. Note: the color value, 0 - 3, must be placed on the Forth stack prior to execution of **PEN**.

Screen 44 is left blank, for growth purposes. Add your own texture words here.

Screen 45 defines the first textured brush stroke, **XBRUSH**. Instead of one value required on the stack (as with **PEN**) two values are needed prior to executing **XBRUSH**. Line 1 stores the colors in the associated variables **CLRA** and **CLRB**. Lines 2 - 15 consist of a loop. Lines 3 - 10 plot the two colors in alternating sequence, flash the brush's position and update the joystick's position for movement. All this is performed **UNTIL** the trigger is pressed (lines 14 and 15).

Screen 46 is left intentionally blank, like screen 44.

```
Scr # 41
  0  ( flashing stick's position  )
  1
  2 : FLASHSTK
  3   STKX @ STKY @ LOC. LOCCLR !
  4   1 COLOR STKX @ STKY @ PLOT
  5   25 MSEC
  6   2 COLOR STKX @ STKY @ PLOT
  7   25 MSEC
  8   3 COLOR STKX @ STKY @ PLOT
  9   25 MSEC
 10   LOCCLR @
 11     COLOR STKX @ STKY @ PLOT
 12   25 MSEC  ;
 13
 14
 15                                ==>

Scr # 42
  0  ( moving tip without marking )
  1
  2 : TIP
  3 BEGIN
  4   FLASHSTK
  5   STKX @ STKY @ OSTKXY
  6   ROT + <ROT + STKX ! STKY !
  7 OSTRIG 0 =
  8 UNTIL  ;
  9
 10
 11
 12
 13
 14
 15                                -->
```

Screens 47 - 48 are similar to each other and to **XBRUSH**, but each produces different color textures: horizontal and vertical, respectively. **HBRUSH** differs from **XBRUSH** in the absence of lines 5 and 9. **VBRUSH** differs from **XBRUSH** in the color used in lines 7 and 9.

Let's Paint

To paint, first load the screen and initialize an Atari graphic mode. Type **ZERO** to place the cursor in the upper-left corner of the screen. Type **TIP** and move it around with the joystick. When finished, press the trigger. Now execute **2 PEN** or another color parameter, and draw on the screen. Again, press the trigger when finished with this mode. By typing **1 3 XBRUSH**, **2 1 VBRUSH** or **2 3 HBRUSH**, you can experiment with the various textures.

Create worlds filled with fantasy and bewilderment. Use the Forth art tools as a springboard, and develop new color patterns. Expand and imagine!

```
Scr # 43
  0  ( drawing with a pen          )
  1
  2 : PEN     ( color ---      )
  3 ( # ) CLRA !
  4 CLRA @ COLOR
  5 BEGIN
  6   STKX @ STKY @ PLOT
  7   FLASHSTK
  8   STKX @ STKY @ OSTKXY
  9   ROT + <ROT + STKX ! STKY !
 10 OSTRIG 0 =
 11 UNTIL  ;
 12
 13
 14
 15                                ==>
Scr # 44
  0
  1
  2   ( INTENTIONALLY LEFT BLANK )
  3
  4       ( FOR FUTURE GROWTH )
  5
  6
  7
  8
  9
 10
 11
 12
 13
 14
 15                                -->
Scr # 45
  0 : XBRUSH  ( clrA clrB ---      )
  1 ( # ) ( # ) SWAP CLRA ! CLRB !
  2 BEGIN
  3   CLRA @ COLOR
  4   STKX @      STKY @      PLOT
  5   CLRB @ COLOR
  6   STKX @ 1 + STKY @      PLOT
  7   CLRB @ COLOR
  8   STKX @      STKY @ 1 + PLOT
  9   CLRA @ COLOR
 10   STKX @ 1 + STKY @ 1 + PLOT
 11   FLASHSTK      STKX @ STKY @
 12   OSTKXY 2 * SWAP 2 * SWAP
 13   ROT + <ROT + STKX ! STKY !
 14 OSTRIG 0 =
 15 UNTIL ;                        ==>
```

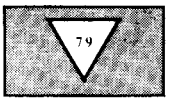
```
Scr # 46
  0
  1
  2    ( INTENTIONALLY LEFT BLANK )
  3
  4        ( FOR FUTURE GROWTH )
  5
  6
  7
  8
  9
 10
 11
 12
 13
 14
 15                                    -->

Scr # 47
  0 : VBRUSH  ( clrA clrB ---      )
  1 ( # ) ( # ) SWAP CLRA ! CLRB !
  2 BEGIN
  3   CLRA @ COLOR
  4   STKX @      STKY @      PLOT
  5   CLRB @ COLOR
  6   STKX @ 1 + STKY @      PLOT
  7   CLRA @ COLOR
  8   STKX @      STKY @ 1 + PLOT
  9   CLRB @ COLOR
 10   STKX @ 1 + STKY @ 1 + PLOT
 11   FLASHSTK      STKX @ STKY @
 12   OSTKXY 2 * SWAP 2 * SWAP
 13   ROT + <ROT + STKX ! STKY !
 14 OSTRIG 0 =
 15 UNTIL ;                           ==>

Scr # 48
  0 : HBRUSH   ( clrA clrB ---     )
  1 ( # ) ( # ) SWAP CLRA ! CLRB !
  2 BEGIN
  3   CLRA @ COLOR
  4   STKX @      STKY @      PLOT
  5
  6   STKX @ 1 + STKY @      PLOT
  7   CLRB @ COLOR
  8   STKX @      STKY @ 1 + PLOT
  9
 10   STKX @ 1 + STKY @ 1 + PLOT
 11   FLASHSTK      STKX @ STKY @
 12   OSTKXY 2 * SWAP 2 * SWAP
 13   ROT + <ROT + STKX ! STKY !
 14 OSTRIG 0 =
 15 UNTIL ;
```

Time-Saving Debugger

Redefining Words



Phil Koopman, Jr.
FPO San Francisco, California

One of the biggest time wasters in writing large Forth programs is the compiling delay that occurs whenever a word defined near the beginning of the dictionary must be changed. The recompilation of the dictionary after a redefinition can often take several minutes for a large application. Numerous methods have been tried to reduce this delay, typically addressing methods to speed up the Forth compiler. Most methods involve large numbers of extra word definitions and significant changes to the dictionary structure of Forth.

This article discusses a simple method to eliminate the time-consuming recompile step after making a minor change. Only one screen of source code is used, and absolutely no changes to the Forth compiler or dictionary structure are required.

The Method

When a small bug is corrected (usually involving the definition of only one word) all that is needed to make the entire program correct is to compile the revised definition and somehow ensure that all references to the old definition are changed to reference the revised definition.

The first way that comes to mind to compile the new word is just to compile it to the end of the dictionary. This will mean that any new word defined will use the revised definition, but no previously defined words will do so. This method fails when the word being revised is used by any previously compiled word.

Another method might be to compile the revised definition directly into the memory used by the old definition in the dictionary. This eliminates all need to change words that use the revised word, since the location of the definition does not change. This method works fine unless the revised definition is larger than the original definition, and therefore will not fit into the dictionary in the old definition's spot.

The solution presented in screen 180 is a combination of the two above methods. The technique used is to define the revised word at the end of the dictionary, then modify the old definition so that it merely executes a jump to the revised definition.

How It Works

The redefinition process is in three steps: using **REDEFINE** to alert the system that a word is to be redefined, redefining the word and then using the word **PATCH** to actually put the new definition into effect.

REDEFINE alerts the system that a word is about to be redefined. It is used in the format

REDEFINE <name>

where <name> is the word name to be redefined. It saves the PFA of the old definition of <name> in the variable **PATCHADDR**.

The second step of the redefinition process is to define the revised definition of <name>. This is usually by means of a **LOAD**, but any means may be used. Note that no special words within the definition are needed, and no editing of screens is required.

The third step of the redefinition process is to use the word **PATCH** to actually patch the old definition to point to the revised definition. **PATCH** is used in the format

PATCH <name>

where <name> is the name of the revised definition. The name used with **PATCH** is usually the same as the name used with **REDEFINE**, but does not have to be. **PATCH** uses the factored word **MAKE-PATCH** to compile the run-time action word <**PATCH**> into the first cell of the old definition's parameter field. The PFA of the revised definition is stored in the second cell of the old definition's parameter field.

At run time, the word <**PATCH**> is executed by the old definition. <**PATCH**> removes the pointer to the old definition from the return stack and replaces it with a pointer to the parameter field of the revised definition. This ensures that any remaining words in the parameter field of the old definition are ignored, and that the return stack is not cluttered up with another return address (in case the revised word uses an unusual exiting technique or is otherwise expecting certain values on the return stack).

Screen 181 shows a very simple test that illustrates the ease of use of this redefinition facility. First, the words **ATEST**, **BTEST** and **TEST** are defined and used. Then the word **ATEST** is redefined and incorporated into the dictionary without having to redefine **BTEST** and **TEST**. Note that the return stack contents are identical for both the old and new versions of **ATEST**.

Limitations

First, this facility is not designed to be very "smart." It will be perfectly happy to crash if the exact sequence of **REDEFINE ... LOAD ... PATCH** is not properly used. Also, it only works for high-level definitions and will not work for code words, constants, variables or the like.

Another limitation is that the word redefined must have at least two cells in its parameter field. This means that a null definition cannot be redefined with this system (a null word is in the form : **NULL** ;). Only one word may be redefined in any **REDEFINE ... LOAD ... PATCH** sequence.

FORGETting the redefined word without also **FORGET**ting the original definition can cause the system to crash.

<**PATCH**> may have to be redefined on systems that pre-increment the IP instead of post-incrementing it. The change is simply:

```
: <PATCH>
R>  2+
@  >R  ;
```

for a sixteen-bit cell size.

On the Bright Side

The words **REDEFINE** and **PATCH** provide a very quick way to change a word definition and examine its results without recompiling the whole dictionary. No change is made in the dictionary structure. When the application is fully debugged (or when a substantial number of bugs have been corrected), the application can be recompiled one time to clean it up and free the extra space used by word redefinitions. Additionally, multiple redefinitions of the same word can be written and quickly tested in the context of the entire system without recompiling.

Summary

The words **REDEFINE** and **PATCH** provide an extremely simple yet effective way to practically eliminate recompile time during debugging. This technique has provided substantial time savings while developing a hotel cash register/control system. The program was developed on ECS MVP/FORTH (a Forth-79 compatible system) running on a well-known 8088-based personal computer system.

Further investigations of redefining words might include greater safeguards against accidental misuse and simultaneous multiple-word redefinitions.

Editor's note: This technique is sometimes referred to as "hotpatching" and must be used very carefully, due to the problems that can arise when the source and object code versions of a program differ.

```
SCREEN #180
 0 \ DEBUGGING PATCH WORDS       P. KOOPMAN JR.  28DEC84
 1 DECIMAL              \ DEFINITIONS IN THE PUBLIC DOMAIN
 2 VARIABLE PATCHADDR    \ PFA FOR REDEFINE, USED BY PATCH
 3
 4 : <PATCH>     ( -> ) ( ACCEPTS IN-LINE PFA FROM DICTIONARY )
 5   R> @    >R ;
 6
 7 : MAKE-PATCH    ( PFA-NEW  PFA-OLD  -> )
 8   ' <PATCH> CFA  OVER !      2+ !  ;
 9
10 : REDEFINE    ( -> )    ( USAGE: REDEFINE name )
11   [COMPILE] '  PATCHADDR ! ;
12
13 : PATCH     ( -> )      ( USAGE: PATCH name )
14   [COMPILE] '    PATCHADDR @    MAKE-PATCH ;
15

SCREEN #181
 0 \ TEST SCREEN FOR PATCHING      PJK   28DEC84   MVP FORTH
 1 DECIMAL
 2 : ATEST  ." TOP OF RETURN STACK="   R@ U.    CR ;
 3 : BTEST  ." BTEST IS AN UNCHANGING WORD IN DEFINITION"  CR ;
 4 : TEST   CR CR   ATEST   BTEST   CR ;
 5
 6 TEST
 7
 8 REDEFINE ATEST
 9 : ATEST  ." THIS IS A REDEFINED ATEST. R@=" R@ U. CR ;
10 PATCH ATEST
11
12 TEST
13
14
15
```

1986 Rochester Forth Conference

June 10–14, 1986
University of Rochester
Rochester, New York

The sixth Rochester Forth Conference will be held at the University of Rochester, and sponsored by the Institute for Applied Forth Research, Inc. The focus will be on Real-Time Artificial Intelligence, Systems and Applications.

Call for Papers

There is a call for papers on the following topics:

- Real-Time Artificial Intelligence
- Forth Applications, including, but not limited to: real-time, business, medical, space-based, laboratory and personal systems; and Forth in silicon.
- Forth Technology, including meta-compilers, finite state machines, control structures, data structures, Forth implementations and hybrid hardware/software systems.

Papers may be presented in either platform or poster sessions. Please submit a 200 word abstract by March 31st, 1986. Papers must be received by April 30th, 1986, and are limited to a maximum of four single spaced, camera-ready pages. Longer papers may be presented at the Conference but should be submitted to the refereed *Journal of Forth Application and Research*.

Abstracts and papers should be sent to the conference chairman: Lawrence P. Forsley, Laboratory for Laser Energetics, 250 East River Road, Rochester, New York 14623. For more information, call or write Ms. Maria Gress, Institute for Applied Forth Research, 478 Thurston Road, Rochester, New York 14619 (716) 235-0168.

A Proposal

Forth Component Libraries

John S. James
Santa Cruz, California

Forth's greatest need is for a clean way to transport large pieces of programs from one developer or installation to another. For example, if you want to support a particular local-area network, or need a file editor, or a quicksort, or a B-tree implementation for a database, you should be able to buy these packages off the shelf. They should run identically on Forth-83 or Forth-79 or on any system from any vendor; and they should be as efficient as hand-coded versions, load in a completely standard way with no "funnies," and never require you to see or know their internals.

This article outlines one approach toward this goal of interchangeable Forth components. It looks not only at sharing code within Forth, but also at having Forth programs cooperate with those written in other languages, such as C. It sketches not only a technology, but also standards and conventions and business planning necessary to get from here to there.

Readers should note that this article presents new work subject to change, not a finished product. We consider only source-code modules; object-code systems such as virtual execution are beyond the scope of this article, although nothing presented here would prevent their use. Readers should also note that this system is not technology-intensive; rather, it uses a little technology to support social conventions and practices, which do the main job.

Technical Basis: Three Legs of a Tripod

The three main technical components of this system are:

1. *I/O redirection and pipes, provided by standard operating systems.*

"Pipes" allow the standard output stream of one program to become the standard input stream of another. Input or output can come from or to another program, or from a file or from the terminal, with no change to the program. Developers can use pipes to string together small programs into useful systems, even if the programs are written in different languages.

UNIX has long used pipes as a major basis for the library of modules and tools which has been largely responsible for its success. More recently, MS-DOS for the IBM PC and compatibles has added redirection and pipes (version 2.0 and greater). This facility can help us not only to modularize Forth programs, but also to use the great body of software already available in other languages. It can support development and testing even when the final product will run in a target environment without an operating system.

Forth offers important advantages within non-Forth environments. Compared to C and other general-purpose languages, Forth allows software developers to get results faster and less expensively. The programs are highly efficient and reliable. Interactive access, fast development speed, and complete control of the system at low or high levels give Forth important advantages for the job of combining the often-incompatible pieces of existing software into useful systems.

2. *A module compiler which eliminates heads not used outside the module.*

To start with the smallest advantage, eliminating heads saves memory without the need for target compilation. It's just not true that because chips are cheap, memory efficiency doesn't count any more. Many developers are still running up against memory limitations, for all sorts of reasons.

This module compiler also helps by reducing the confusion caused by too many names in the dictionary. When a program reaches the size of a thousand words or so, it becomes hard to keep track of them all, and hard to read listings of software which might use almost any of them. Using modules cleans up glossaries by eliminating most of these words, helps standardize some of the words which do remain, and makes listings more readable by controlling the proliferation of names which programmers must remember.

But most important of all, removing unwanted heads forces developers to think through exactly what facilities they want to provide to their users. They must offer a complete, explicit set of capabilities for some purpose, since users will not normally make any changes to the source or even look at it. No longer can programmers get away with half-baked software which requires tinkering to do anything useful, or even to load on someone else's system.

The module compiler separates the interface to the world from anything that happens inside. The interface must be fully documented in a spec sheet for the module. Usually this spec sheet will be identical for all CPUs, and for all versions of Forth and all vendors' systems. But within each implementation, the developer has complete freedom to use all facilities available to get the most efficient code, as long as the results match the specifications.

Modules can call other modules, of course. Any such dependencies must be documented in the spec sheet.

3. *Generic software design when appropriate.*

Forth encourages "generic" modules which can operate on any data structures. For example, a generic quicksort can take as arguments the address of a Compare operation, the address of an Exchange operation, and the number of items to be sorted. It can then sort any data whatever on which these operations can be defined, with no change at all to the program. The generic design completely separates the quicksort logic from everything else — the data structures, the operations on them, memory management, and so on.

This particular example, the generic quicksort, is good enough for most sorting tasks on randomly-accessed data. (MS-DOS and other operating systems already have sort utilities for text files with variable-length records, for which it would be hard to define efficient Compare and Exchange operations.) The Forth quicksort can handle any data types in key fields and subfields; sort in ascending, descending, or mixed sequences; and sort data on disk as well as in memory. Thanks to the module compiler, this useful facility only adds one word, **SORT**, to the dictionary.

Overcoming the Dialect Problem

Despite the Forth-83 Standard, the fact is that not all Forths are the same. And since the standard doesn't cover everything, there can be incompatibilities even within it.

Of course, the module library cannot abolish these difficulties. But it can be independent of them in every major way. Each important module in the library will have to be modified for each different Forth system: Forth-83 and Forth-79 and, for optimum performance, for each vendor's system and CPU. Usually only minor changes will be required; the developer of the module, the Forth system vendor, or a user can make them. The library, which will exist as a set of files, will have a different version for each supported Forth system; whoever orders a module (or the whole library) will need to specify which Forth they want it for, admittedly an inconvenience.

But what counts more is that almost all modules will have exactly the same behavior across all different Forths. Each module will have only one spec sheet, identical for all kinds of Forth, blind to all versions or variations. The module compiler separates the interface from the insides of the module; it lets developers hold the interface constant, while at the same time coding the internals for maximum efficiency.

The internal coding of a module need not follow any standard, even if the module is to run on standard systems. Instead, the developer can do anything for performance — including use of code, of course. It may be convenient to follow the standard when feasible, in order to ease or eliminate any changes required to make the module run on different vendors' implementations of the standard. It may also be convenient to define a standardized library of code modules as they are needed, so that most other modules will not need to use any code and can be CPU independent without loss of performance.

On Separate Compilation

Forth provides separate compilation of program components in a way different from most languages. Normally, a compiler produces relocatable modules, and a linker combines them into an object module. Separate compilation saves compile time, enforces separation between a module's interface and its internals, and allows modules written in different languages to be combined into one program.

Forth with a module library provides these advantages in a different way, without the inconvenience and overhead of linking relocatable modules. A compiled Forth module does not become a relocatable file, but instead it becomes part of the system itself and can be saved with the system, to be loaded automatically with it in the future. The disadvantage is that you cannot rearrange modules into a new system without going back to the source code. The advantage is that the system which includes the compiled module can be an interactive environment which can continue to grow.

Programs written in different languages can be combined in several ways. Code subroutines in Forth have always been common. Pipes and command files (batch files) can combine separate programs written in different languages. And Forth itself can look like an ordinary assembly-language program to the operating system and call subroutines written in other languages — not yet a common practice, but sometimes a particularly valuable one because it can give interactive access to software not otherwise interactive. I believe it will be more productive to use standard linkers to combine Forth with software components written in other languages, rather than providing separate compilation of Forth modules into relocatable object programs.

The Library in Business: Getting from Here to There

Forth differs from other programming languages in that it has only rarely had significant institutional backing, either from academic or commercial institutions, but has kept growing on its own. Much of the existing body of work has been contributed to the public domain, often by computer professionals working on Forth as a sideline or by academics in departments other than computer science. Often this work has good quality inside, but lacks full development as a product. Hence the hobbyist image which has hindered Forth; hobbyists don't mind chewing on unfinished work, because they don't put a price on their time.

Meanwhile, vendors find it hard to make much profit on the sale of Forth systems. Many users who would gladly pay for productization and support choose public-domain systems instead, in order to get complete source code and control of their tools, with no strings attached. The lack of profit to vendors further impedes the development of polished, easily-used tools, completing a vicious circle which has slowed the development and acceptance of Forth.

A module library system can help us break out of this dilemma. It allows developers to sell application support to a larger potential market than otherwise, because their work can run uniformly and with no tinkering on anybody's Forth system. It allows vendors to compete in the tools and components for building applications, instead of competing either in raw Forth

systems or in finished end-user products, which is not the right business for everyone.

I'm leaning toward the following three-tier pricing structure for a software-component library. The module compiler itself (about five screens of source code), and perhaps some basic support for pipes and I/O redirection, should be public domain, so that vendors can distribute it freely to their existing and future customers. These parts of the library will need some systems work to interface them to existing Forth implementations. The vendors are in the best position to maintain that code, so it should be as easy as possible for them to do so. Forth vendors usually own all parts of their systems; few would change this policy and build in a piece of code owned by someone else.

At the next price level would be a package of short, immediately-useful modules such as a quicksort, string handling, good file-and-directory support, and general interface to other operating-system facilities such as clock/calendar, or obtaining multiple arguments from the program's command line. These modules will handle the oddities of the operating system, so that users will not need to know about them. This package of routines would be low priced, perhaps in the fifty to a hundred dollar range to the end user and with no charge for distributing object code, so that it would be available to hobbyists and also as an out-of-pocket purchase by professionals not yet able to justify a significant expense to their managements. It could be licensed and distributed by vendors, and would usually serve as the introductory package to the library. Forth might develop a standard library of tools available across different implementations, like the standard library of C.

At the high-price end would be professional, application-oriented modules such as support for a particular local-area network, an optimized B-tree or relational database-access system, or comprehensive support for the special features of a new computer. Each of these modules might cost several hundred dollars, with licensing available for unlimited site use, or for per-copy or unlimited distribution of object code.

Potential customers for these packages will know what they want, have a commercial need for it, be able to get complete information from the spec sheet, and be able to try out the software at a reasonable cost. Note that anyone could develop and sell these modules directly to users, with or without going through Forth system vendors or any central library administration.

If you have any suggestions and/or want to be included on a mailing list for news about this system as it develops, write to me at P.O. Box 486, Santa Cruz, California 95061 USA.

Advertiser's Index

1985 Forth National Convention

The Forth Interest Group held its seventh annual National Convention on September 20-21, 1985. The theme of the event was "The Forth Elements: Earth, Aerospace, Fire and Water." Of particular interest to the more than five hundred attendees were presentations focusing on the current use of Forth in space exploration and research, and in fusion technology.

The convention began with a panel whose members outlined the exciting uses of Forth in the aerospace industry. Elizabeth Rather of Forth Inc. first approached the FIG program committee with the idea for this panel, then coordinated the speakers with the assistance of Mary Lindsay and chaired the very successful session. Interestingly, a large percentage of the audience had used Forth professionally within the aerospace industry.

Robert Wood spoke at some length during the aerospace portion of the program about his training as a space shuttle "programmer in space." Forth's interactive nature and usefulness in process control has been key to the successful completion of certain experiments on past shuttle missions, and the mission on which he is scheduled to fly will benefit from the immediacy of having a Forth programmer in orbit with the experiments. While the exact nature of the mission is confidential, positive results are expected to make financially feasible the production of specific substances too costly to obtain in quantity on the earth's surface. We will be counting down with Robert this Spring, and wish him a successful flight.

Fellow panel members included Henry Harris, a missions design and operations manager at the Pasadena-based Jet Propulsion Laboratory, and Greg Schmidt, whose Forth expertise has been put to use on the space telescope. Mr. Harris, who also spoke at this year's Rochester Forth Conference, detailed a ground-based operation controlling instrumentation in the space shuttle's cargo bay, principally the Shuttle Imaging Radar. The project used Forth on six IBM XTs linked via Ethernet to provide integrated control of the mission. The speaker credited Forth as essential to operating in demanding conditions and adjusting quickly to critical hardware failures, and pointed to the success of the missions as a testimony to Forth. When the future brings a manned scientific platform in space, its roots will at least partly lie in this important work.

Paul Heckel, president of QuickView Systems and author of *The Elements of Friendly Software Design*, intrigued convention attendees with the new software metaphor he has developed. His "Zoomracks" concept attempts to bring to the microcomputer industry an infusion of vitality much like that created by the "electronic spreadsheet" metaphor some years ago. Like many fundamentally new concepts, this one lures the casual observer to get hands-on experience, to develop a "feel" for the idea's utility.

A featured event of every year's National Convention is the evening banquet. This year's keynote speaker was Lawrence P. Forsley, editor-in-chief of *The Journal of Forth Application and Research* and computer systems group leader at the Laboratory for Laser Energetics, where he and the staff have used Forth for over nine years. He spoke at length on the control of a twenty-four beam laser used in fusion research, and showed how Forth can be managed on large-scale projects. He went on to address Forth's viability. Larry pointed out, in particular, that Forth's continued popularization in coming years will depend to a great extent on its penetration into academic settings. Others have echoed his sentiment that a full-featured Forth package should be made available to colleges, in particular to engineering departments, at a reasonable price. Forth has always excelled in such environments (witness Stanford University's engineering course using Forth) but it will require a concerted effort to reach the needed scale.

At the same banquet, FIG Secretary Kim R. Harris presented the annual FIGGY award. Previous recipients of the award selected Thea Martin as the individual whose volunteer work on behalf of the Forth community during the year most deserve FIG's recognition and gratitude. Thea is the publisher of *The Journal of Forth Application and Research* and serves as a member of the board of directors of the Forth Interest Group.

In addition to the well-attended lecture series, special events at this year's convention brought together special users groups (e.g., F83, Forth Inc. and Mountain View Press), and small group discussions about FIG Chapters, Forth vendors and establishing Forth as a special interest group on a large-scale commercial data-base service. Frequent tutorials, vendor exhibits and hours of informal networking and technical discussions rounded out the two-day affair.

—Marlin Ouverson

FIG Chapters

U.S.

• ALABAMA

Huntsville FIG Chapter
Call Tom Konantz
205/881-6483

• ALASKA

mbKodiak Area Chapter
Call Horace Simmons
907/486-5049

• 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

• ARKANSAS

Central Arkansas Chapter
Twice Monthly, 2nd Sat., 2p.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., Room A
Call Tom Ghormley
916/444-7775

Bay Area Chapter
Silicon Valley Chapter
Monthly, 4th Sat.
FORML 10 a.m., Fig 1 p.m.
ABC Christian School Aud.
Dartmouth & San Carlos Ave.
San Carlos
Call John Hall 415/532-1115
or call the FIG Hotline:
408/277-0668

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

Southeast Florida Chapter
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

Cache Forth Chapter
Call Clyde W. Phillips, Jr.
Oak Park
312/386-3147

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

Central Iowa FIG Chapter
Call Rodrick A. Eldridge
515/294-5659

Fairfield FIG Chapter
Monthly, 4th day, 8:15 p.m.
Call Gurdy Leete
515/472-7077

• 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
504/899-8922

• 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

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

St. Louis Chapter
Monthly, 1st Tues., 7 p.m.
Thornhill Branch Library
Contact Robert Washam
91 Weis Dr.
Ellisville, MO 63011

• 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

Rockland County Chapter
Call Elizabeth Gormley
Pearl River
914/735-8967

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, 4th Sat., 10 a.m.
Drexel University, Stratton Hall
Call Melonie Hoag
215/895-2628

• 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

mbAustin 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

Periman Basin Chapter
Call Carl Bryson
Odessa
915/337-8994

• UTAH

North Orem FIG Chapter
Contact Ron Tanner
748 N. 1340 W.
Orem, UT 84057

• VERMONT

Vermont Chapter
Monthly, 3rd Mon., 7:30 p.m.
Vergennes Union High School
Rm. 210, Monkton Rd.
Vergennes, VT
Call Don VanSyckel
802/388-6698

• VIRGINIA

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

Potomac Chapter
Monthly, 2nd Tues., 7 p.m.
Lee Center
Lee Highway at Lexington St.
Arlington, VA
Call Joel Shprentz
703/860-9260

Richmond Forth Group
Monthly, 2nd Wed., 7 p.m.
154 Business School
Univ. of Richmond
Call Donald A. Full
804/739-3623

• WISCONSIN

Lake Superior FIG Chapter
Call Allen Anway
715/394-8360

MAD Apple Chapter
Contact Bill Horzon
129 S. Yellowstone
Madison, WI 53705

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
D.J. Neale
58 Woodland Way
Morden, Surry SM4 4DS

• FRANCE

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

• GERMANY

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

• HOLLAND

Holland Chapter
Contact: Adriaan van Roosmalen
Heusden Houtsestraat 134
4817 We Breda
31 76 713104

FIG des Alpes Chapter
Contact: Georges Seibel
19 Rue des Hirondelles
74000Annely
50 57 0280

• 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

• JAPAN

Japan Chapter
Contact Toshi Inoue
Dept. of Mineral Dev. Eng.
University of Tokyo
7-3-1 Hongo, Bunkyo 113
812-2111 ext. 7073

• 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

NOW AVAILABLE

20^{00}

FROM THE FORTH INTEREST GROUP

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