$10 Volume XIII, Number 4 November 1991 December

# FORTH DIMENSIONS

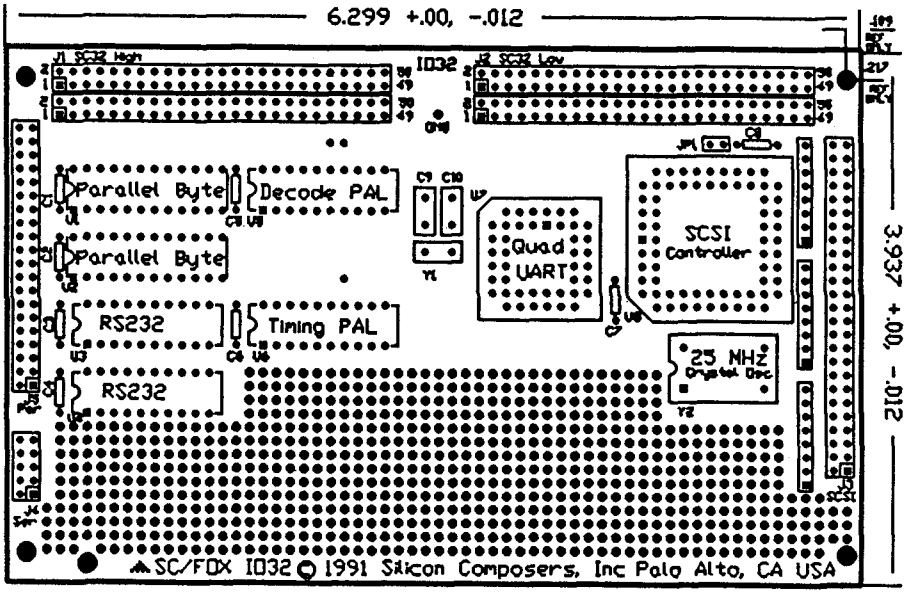
**Combsort in Forth**

**QuikFind String Search**

**New Stack Tools**

# SILICON COMPOSERS INC

## *Announcing the SC/FOX IO32 Board for FAST* Forth I/O




**SC/FOX IO32 Board Features**

- The IO32 is a plug-on daughter board for either the SBC32 stand-alone or PCS32 PC plug-in single board computers.
- 5 MB/sec SCSI Port.
- Attach up to 7 SCSI Devices.
- 4 RS232 Serial Ports, up to 230K baud.
- 16-bit Bidirectional-Parallel Port, may be used as two 8-bit ports.
- 2 programmable counter/timers.
- Prototyping area on board.
- All bus signal brought out to pads.
- Full Interrupt Support.
- Two 50-pin user application connectors.
- No jumpers, totally software configurable.
- Driver software source included.
- Single +5 Volt low-power operation.
- Full ground and power plane.
- 4 Layer, Eurocard-size: 100mm x 160mm.
- User manual and interface schematics included.
- Low chip count (8 ICs) for maximum reliability.
- Test routines for SCSI, parallel, and serial ports supplied in source code form.
- Plug together up to 6 IO32 Boards in a stack.

**Fast Data-Dispersion Program Example**

The program, SEND below, reads 1K blocks from a SCSI drive and transmits them out one of the IO32 board's four RS232 serial ports at 230K Baud. SEND uses only IO32 facilities. Disk read speed is limited by SCSI drive speed.

**Program Example**

```
CREATE BUFR  2560 ALLOT   ( 10k disk buffer)
: PUT ( #k)               ( 1KB blocks to serial)
  1024 * BUFR BYTE +      ( end of buffer)
  BUFR BYTE DO            ( start of buffer DO)
   I C@                   ( get next character)
   UEMIT                  ( and emit via serial)
   LOOP ;                 ( until done)
: SEND ( block# #k)       ( send 1K blks to serial)
  230KB                   ( baud rate=230KBaud!)
  BEGIN ?DUP WHILE        ( while blocks remain..)
    2DUP 10 MIN           ( max 10K in buf)
    >R  BUFR R@ SCSIRD    ( read nK from SCSI)
    R@ PUT                ( and put to serial)
    R@ -                  ( decrement remaining)
    SWAP R> + SWAP        ( up new starting block)
    REPEAT                ( repeat remaining test)
  DROP ;                  ( discard blk# and exit)
```

For additional product and pricing information, please contact us at:
**SILICON COMPOSERS INC 208 California Avenue, Palo Alto, CA 94306 (415) 322-8763**

# Contents

## Features



## Departments

# Guest Editorial:
# How You Can Help

*We've been talking to ourselves for too long, and we need to talk to the rest of the world. While I am off setting up referees for* FD*'s object-oriented programming contest (we have quite a few exciting entries), Horace Simmons offers the following important guest editorial.*

*A FIG Chapter leader who immigrated to the San Francisco Bay area, Horace took the initiative to involve himself in FIG's affairs. He has provided valuable insight and ideas at quite a few meetings of FIG's Business Group.*

*Please take this guest editorial to heart, discuss it at your chapter meetings, and, most of all, act on it!*

*—Editor*

FIG exists to provide a structure for Forth programmers to communicate with each other about Forth and with those who wish to learn more about the language. For several years, FIG has been more successful with the dialogue with its members than it has been with those outside its organization. Use of Forth has continued to grow over the years, even though the growth has been outside the ranks of full-time, professional programmers and hobbyists. *EDN*'s editor reports that 10% of its 100,000 readers use Forth. Because FIG's membership is not that large, we know that most of those readers cannot make use of FIG's services. We also know that FIG has not been reaching them with information about how to network with other Forth programmers. Now, FIG could spend some of its revenue to run advertisements in *EDN* to try to reach those users. Or, some FIG member could write an article about one of his projects and send it to *EDN*.

When *EDN* publishes that article, the member who wrote it makes some money. Assuming that the member mentioned how Forth was used and how it contributed to the success of the project, Forth users will be reached and middle-level management can be influenced. By including a footnote mentioning FIG, or a bibliographic reference to FIG (P.O.Box 8231, San Jose, California 95155, 408-277-0668, fax 408-286-8988) and, perhaps, to the vendor of the Forth package, anyone reading the article can re-

## dpANS Forth Released for Public Review!

*Major milestone*—the Draft Proposed ANS Programming Language Forth was to enter its official public review period in October. Copies of the proposed standard may differ from development versions (i.e., the "BASIS" documents), and can be purchased from Global Engineering Documents, Inc., 2805 McGaw Avenue, Irvine, California 92714. Ask for document #X3.215-199x. From within the United States and Canada, call 800-854-7179; from other countries, call 714-261-1455. The U.S. price is $50 per copy; for international orders, the price is $65 per copy. *[This data is from a notice posted 9-18-91 on GEnie by the chairman of the X3J14 committee. However, Global Engineering had not received the document as of 9-30-91, and their spokesman informed* FD *that pricing may be subject to change.]*

The public-review period extends from October 18, 1991 through February 25, 1992. Please send all comments to X3 Secretariat/CBEMA, Att'n: Lynn Barra, 311 First Street N.W., Suite 500, Washington D.C. 20001-2178. Send a copy of all comments to American National Standards Institute, Att'n: BSR Center, 11 West 42nd Street, New York, New York 10036.

**Forth Dimensions**
Volume XIII, Number 4
November 1991 December

Published by the
**Forth Interest Group**

*Editor*
Marlin Ouverson

*Circulation/Order Desk*
Anna Brereton

*Forth Dimensions* welcomes editorial material, letters to the editor, and comments from its readers. No responsibility is assumed for accuracy of submissions.

Subscription to *Forth Dimensions* is included with membership in the Forth Interest Group at $40 per year ($52 overseas air). For membership, change of address, and to submit items for publication, the address is: Forth Interest Group, P.O. Box 8231, San Jose, California 95155. Administrative offices: 408-277-0668. Fax: 408-286-8988. Advertising sales: 805-946-2272.

Copyright © 1991 by Forth Interest Group, Inc. The material contained in this periodical (but not the code) is copyrighted by the individual authors of the articles and by Forth Interest Group, Inc., respectively. Any reproduction or use of this periodical as it is compiled or the articles, except reproductions for non-commercial purposes, without the written permission of Forth Interest Group, Inc. is a violation of the Copyright Laws. Any code bearing a copyright notice, however, can be used only with permission of the copyright holder.

*The Forth Interest Group*

The Forth Interest Group is the association of programmers, managers, and engineers who create practical, Forth-based solutions to real-world needs. Many research hardware and software designs that will advance the general state of the art. FIG provides a climate of intellectual exchange and benefits intended to assist each of its members. Publications, conferences, seminars, telecommunications, and area chapter meetings are among its activities.

"*Forth Dimensions* (ISSN 0884-0822) is published bimonthly for $40/46/52 per year by the Forth Interest Group, 1330 S. Bascom Ave., Suite D, San Jose, CA 95128. Second-class postage paid at San Jose, CA. POSTMASTER: Send address changes to *Forth Dimensions*, P.O. Box 8231, San Jose, CA 95155."

ceive a pointer to how he or she can personally benefit.

A hundred of our membership identified *EDN* as a regular trade publication they read. As each of you in turn publishes one article, think of the impact, of the "mind space" created among project managers. Think of the new users brought to FIG, made aware of the extensive library of Forth materials, introduced to the FORML conference and the Rochester conference. Think of the extra money, the prestige, the item on your resume, as you do your part.

*EDN* is just one of 200 magazines identified in our member survey. If *EDN* is not your magazine, why not write for *Chemical and Engineering News*, or *Automotive Engineering*, or the *Journal of the American Ceramics Society*. All you have to do to help some of your colleagues is write an article for your own area of expertise and submit it for publication to a journal which you read. The article need not and, indeed, should not be an article about Forth. Just mention in it how the software which enabled your success was written in Forth. Include, perhaps, just three or four lines of straightforward code that might be readable by those knowledgeable about your subject, even if they don't use Forth. If that doesn't seem feasible, don't include any code. You are a successful practitioner in your field; others will want to benefit from your experiences and your judgment.

Many examples of this kind of article abound. The May/June 1991 issue of *Computers in Physics* has an article entitled "A General Purpose Interactive Programmable Laboratory Interface System Using the IEEE-488 Bus" by B. D. Hall of Lausanne, Switzerland. It is almost five pages of material on how to implement a distributed, interactive instrument control structure for use in a physics research lab. While the article is about controlling instruments, the message is how adaptive and effective Forth is for scientists in the lab.

*Sensors Magazine*, April, 1991, has a feature article entitled "Environmental Control in Three Dimensions" by Edward K. Conklin of Forth, Inc. The article describes the design requirements, and the hardware required to control temperature, pressure, and humidity in the General Motors subsidiary Saturn Corporation automotive manufacturing complex in Tennessee. Forth and Forth, Inc. are mentioned several times in the article, including a sidebar on Forth for industrial control. Readers are exposed to the strengths of Forth, without a single line of code being published.

Perhaps you remember how you came to be introduced to Forth and how, in the beginning, you relied on others for help and encouragement. Now you are in a position to repay that debt—not to the one who brought you to Forth, but to someone else who is ready and needs the same help you did.

*—Horace O. Simmons*

# Letters

*Letters to the Editor—and to your fellow readers—are always welcome. Respond to articles, describe your latest projects, ask for input, advise the Forth community, or simply share a recent insight. Code is also welcome, but is optional. Letters may be edited for clarity and length. We want to hear from you!*

**Anti-Vendor Bias?**

Dear Sir:

I would like to correct a misrepresentation of our product by Frank Sergeant in "An Introduction to Pygmy Forth (*FD* XIII/2). Mr. Sergeant insinuates that HS/FORTH does not compile as fast as its 40,000 line-per-minute advertising claim indicates, and that Pygmy Forth would be just as fast if only he would play the same tricks with his numbers. It is obvious that his comment refers to HS/FORTH since only we make that claim. Had he been interested in facts rather than just an opportunity to promote his product, he could have easily asked us for the details. (A '286, not a '486 as suggested, no blank lines, many words per line, 80-character lines, not little 64-character ones, twice as fast as Pygmy Forth, both so much faster than anything else it doesn't matter anyway). I also notice that his benchmark applies to a pygmy application in a pygmy system, the figures would not necessarily hold for a large application in a large system. Ours is for a large application in a large system. Our installable/removable hash system has also been used reliably for several years now, and will no doubt be the unacknowledged inspiration of many other "improved" Forth systems. Copying ideas developed by others may be a form of flattery; falsely denigrating those original products to flatter the copy is pretty low.

It is regrettable that the anti-vendor, pro-freebie bias of *Forth Dimensions* allows such articles to be published. Such a contentless article by *any* vendor about his product would have been rejected immediately. As a matter of history, *Forth Dimensions* doesn't publish information about any vendor's product or features except as paid advertising. Since other magazines publish very little on Forth, this policy effectively prevents the discussion of the relative merits of vendor systems, and restricts editorial coverage to consultants and hobbyists, who often "invent" features already in commercial systems. Advertising, however, comes in all forms. This article acts as advertising for Mr. Sergeant's consulting business. Donate a minimal Forth system free, get free advertising in *Forth Dimensions* and on the BBS's, then pick up the bucks consulting and selling utilities. A popular route with too many Forth hackers, and

*(Continued on page 10.)*

# Combsort in Forth

*Walter J. Rottenkolber*
*Visalia, California*

In their article, "A Fast, Easy Sort" (*BYTE*, April 1991), Richard Box and Stephen Lacey describe how, by adding three lines of code to the ubiquitous bubble sort, they created Combsort, a fire-breathing monster capable of a scorching 2600% increase in sorting speed. This seemed too good to be true; it also was the April issue. But read on.

To test the claims made of Combsort, I decided to use the routines published in "The Challenge of Sorts" (*FD* XI/3). These provide for an integer array that can be filled with eight different patterns of data. A comprehensive analysis section is included, but I had to forego it, as my computer—a 5 MHz Kaypro II—doesn't have a built-in clock. All times are by the Armstrong method, i.e., me staring bleary-eyed at my watch.

The screens provide Forth code for the data array and patterns from the Challenge. If you have a fast computer with a built-in clock, you will be much happier with the original test suite, as it automates the entire sort test and prints a comprehensive report of the results.

**Now is the time to retire your bubble sort to the museum of archaic algorithms!**

BUBBLE1 is the Forth translation of the True BASIC listing that provides the basis for Combsort. This version uses a flag (-SWITCH) to check for the clean pass that marks the end of the sort.

COMB1 is the Combsort derived from BUBBLE1. Only three lines of code make the difference. These introduce a gap between the elements to be compared. As the sort progresses, the gap narrows, step by step, to one, at which point the Combsort behaves like a bubble sort. The initial gap is calculated by dividing the array size by a "shrink factor," whose value is 1.3, and converting the result to an integer. In Forth, the scaling routine (10 13 */) does the calculation. At each cycle in the sort, the gap is narrowed by the same factor. Cox and Lacey found the shrink factor by trial and error. Too small, and the sort behaves more like a bubble sort; too large, and the sort becomes chaotic, varying in speed unpredictably with minute changes in array size.

COMB2 is my version of their optimized Combsort11. If you take an integer and divide repeatedly by 1.3, as in the gap calculation, eventually the progression will pass through the values nine, ten, or 11. Cox and Lacey determined that the gap sequence following nine and ten sorts more slowly than the sequence beginning with 11. So they added a switch statement (they wrote this in C) to trap the nine and ten gaps and convert them to gap 11.

You will find the QUICK and BUBBLE2 sort routines used in the tests in the Challenge article.

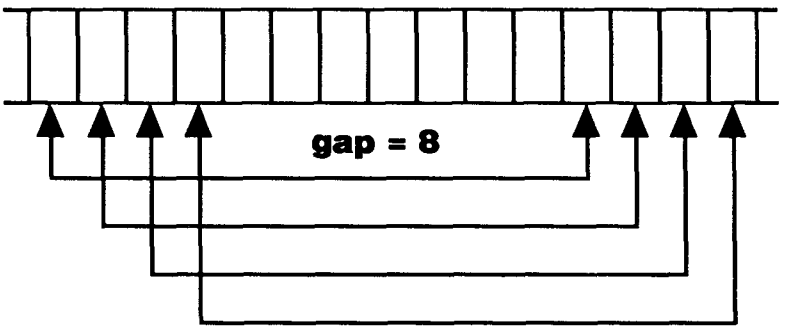
The bubble sort sequence proceeds by repeatedly sweeping an array from one end to the other. Step by step, it compares two adjacent elements in the array and, depending on the outcome, may swap them. Values moving in the direction of the sweep can make several steps toward their sorted location. However, values that must move against the sweep do so only one step at a time. To speed up the sort, a way must be found to gather these slow values and bring them rapidly to the head of the sweep.

The Combsort takes a direct approach. It simply inserts a gap between the elements and then does a bubble sort. As a result, the sort starts at both ends of the array (see Figure One). This pumps the slow values from the "wrong" end of the array to where they belong. As the gap narrows, the center of the array is included in the sort, but the leap-frog action of the sort persists until the gap narrows to one.

The sort times are shown in Figure Two. The sort patterns are as follows:

Walter J. Rottenkolber says that Forth provides the same close-to-the-silicon feel as assembler, but without the pain. Early on, he experimented with fig-FORTH and other languages, but still

**Figure One.** Combsort sweeps data from both ends.




Ramp—ascending values, already sorted.
Slope—descending values.
Wild—random signed values.
Shuffle—a Ramp randomly reordered (no duplicates).
Byte—random eight-bit values.
Flat—a single random value.
Checker—two random values placed alternately on even/odd addresses.
Hump—Gaussian distribution of values.

**Figure Two.** Comparative sort times on test data (on a 5 MHz Kaypro II).

| | **Sort times** (seconds) | | | | |
|---|---|---|---|---|---|
| **Data pattern** | COMB1 | COMB2 | QUICK | BUBBLE1 | BUBBLE2 |
| Ramp | 35 | 38 | 13 | 2 | ----- |
| Slope | 41 | 42 | 14 | 3150 | 2078 |
| Wild | 53 | 52 | 22 | | |
| Shuffle | 52 | 52 | 22 | | |
| Byte | 48 | 50 | 22 | | |
| Flat | 36 | 38 | 22 | | |
| Checker | 37 | 40 | 22 | | |
| Hump | 47 | 47 | 22 | | |

When I first ran BUBBLE2 on the Slope data pattern, I thought my computer died and went to heaven. After spending the better part of a day trying to debug the sort code and reviewing all about nested DO...LOOPs, I concluded that the sort actually was working...and working... all 2078 seconds of it. Then BUBBLE1 took a glacial 3150 seconds (that's 52+ minutes, Bubba) to sort the same pattern. This ended any notion to test the bubble sorts further.

The Combsort gave an amazing account of itself. It is 7583% faster than the bubble sort on which it is based, and an average of only 54% slower than the Quicksort. Because of its design, it spends a somewhat greater time than the other sorts on data that is already sorted or of flat value. I regard this as a small price to pay for such a simple high-performance sort routine.

I found no advantage to the optimized Combsort. On my system, it actually ran about 5% slower than the simpler version.

All the sort times should be considered as relative and not absolute. You can boost the performance by revising S@, S!, COMPARE, and EXCHANGE. These words were hampered by extra code used for the test programs. In running the time tests, I left them as-is because BUBBLE2 and Quicksort used them. I removed some of these extra words when cleaning up the screens for this article, and discovered that the times were now cut in half.

To sum up, the Combsort is real. If you have been using a bubble sort, now is the time to retire it to your museum of archaic algorithms. If you are using a complex sort because nothing else was fast enough, check out the Combsort. I'm quite impressed at what a clever idea and three lines of code can do, and you will be too.

*(Code begins on next page.)*


# Total control with *LMI FORTH*™

***For Programming Professionals: an expanding family of compatible, high-performance, compilers for microcomputers***

**For Development:**
Interactive Forth-83 Interpreter/Compilers for MS-DOS, OS/2, and the 80386
- 16-bit and 32-bit implementations
- Full screen editor and assembler
- Uses standard operating system files
- 500 page manual written in plain English
- Support for graphics,floating point, native code generation

**For Applications:** Forth-83 Metacompiler
- Unique table-driven multi-pass Forth compiler
- Compiles compact ROMable or disk-based applications
- Excellent error handling
- Produces headerless code, compiles from intermediate states, and performs conditional compilation
- Cross-compiles to 8080, Z-80, 8088, 68000, 6502, 8051, 8096, 1802, 6303, 6809, 68HC11,34010, V25, RTX-2000
- No license fee or royalty for compiled applications

*Laboratory Microsystems Incorporated*
*Post Office Box 10430, Marina del Rey, CA 90295*
*Phone Credit Card Orders to: (213) 306-7412*
*FAX: (213) 301-0761*

```
      0
0 \   Combsort
1 \S
2           Combsort in Forth
3
4
5         Routines to Test Sort
6     From Forth Dimensions Vol.XI, No.3
7           Sept/Oct 1989
8
9     "The Challenge of Sorts", p.24-29
10
11
12 Walter J. Rottenkolber
13
14
15
```

```
      1
0 \  Combsort Load Screen
1
2 2  8  THRU
3
4
5
6
7
8
9
10
11
12
13
14
15
```

```
      2
0 \  Data Array and Utilities
1 :  CELLS  ( a -- a' )  2* ;
2 : 2CELLS  ( a -- a' )  2* 2* ;
3
4 1024 CONSTANT  ITEMS
5 CREATE  DATA  ( -- a )  ITEMS CELLS ALLOT ;
6
7 : S@ ( index -- n )  CELLS DATA + @ ;
8 : S!  ( n index -- )  CELLS DATA + ! ;
9
10 : COMPARE  ( n1 n2 -- -1 | 0 | 1 )
11     2DUP ( )R ) 1 AND R) OR ;
12
13 : EXCHANGE  ( #1 #2 -- )
14     2DUP S@ SWAP S@  ROT S! SWAP S! ;
15
```

```
      3
WJR07MAY91 \  Random Number Generator
           VARIABLE  SEED
           : SETUP  ( -- )  1234 SEED ! ;

           : RANDOM  ( -- n )
              SEED @  31421 * 6927 + DUP SEED ! ;

           : CHOOSE  ( limit -- 0..limit-1 )
              RANDOM  UM*  SWAP DROP ;

           : GAUSS  ( n -- u )
              RANDOM  0      RANDOM  0  D+  RANDOM  0  D+
              RANDOM  0  D+  RANDOM  0  D+  RANDOM  0  D+
              6 UM/MOD  SWAP DROP UM*  SWAP  DROP ;
```

```
      4
WJR07MAY91 \  Random Data Patterns

           : RAMP  ( -- )  ITEMS 0 DO  I  I  S!  LOOP ;
           : SLOPE  ( -- )  ITEMS 0 DO ITEMS 1- I - I  S!  LOOP ;
           : WILD  ( -- )  ITEMS 0 DO  RANDOM  I  S!  LOOP ;
           : SHUFFLE  ( -- )
              RAMP ITEMS 0 DO ITEMS CHOOSE  I  EXCHANGE LOOP ;
           : BYTE  ( -- )  ITEMS 0 DO  256 CHOOSE  I  S!  LOOP ;
           : FLAT  ( -- )  RANDOM ITEMS 0 DO DUP  I  S!  LOOP DROP ;
           : CHECKER  ( -- )  RANDOM  RANDOM
              ITEMS 0 DO DUP  I  S!  SWAP LOOP  2DROP ;
           : HUMP  ( -- )  ITEMS 0 DO  256 GAUSS  I S!  LOOP ;
```

```
      5
WJR07MAY91 \  Data sort test

           : TEST-DATA  ( -- )
              \ Checks if data is sorted.
              DATA @  ITEMS 1 DO  DATA I CELLS + @ SWAP OVER )
              ABORT" Data has not been sorted."
              LOOP DROP ;
```

United States Postal Service

**Statement of Ownership, Management and Circulation**

1) Title of Publication: Forth Dimensions
Publication number: U.S.P.S. 002-191
2) Date of Filing: 9/5/91
3) Frequency of Issue: Bi-Monthly
No. of issues published annually: 6
Annual subscription price: $34/40/46
4) Location of known office of publication:
1330 S. Bascom Ave., Suite D
San Jose, Santa Clara County, California 95128-4502
5) Location of the headquarters or general business offices of the publisher: Same as above
6) Publisher:
Forth Interest Group, Inc.
P.O. Box 8231
San Jose, California 95155
Editor: Marlin Ouverson
Same as above
7) Owner: Forth Interest Group, Inc.
P.O. Box 8231
San Jose, California 95155
8) Known bondholders, mortgagees, and other security holders owning or holding 1% or more total amount of bonds, mortgages, and other securities: none
9) The purpose, function and non-profit status of this organization and the exempt status for Federal Income Tax purposes have not changed during the preceding 12 months.
10) Extent and nature of circulation

| | Avg. # copies/issue during preceding 12 mos. | Actual # copies of of single issue nearest to filing date |
|---|---|---|
| A. Total no. copies printed: | 1934 | 1750 |
| B. Paid/requested circulation: | | |
| 1. Sales: | 13 | 10 |
| 2. Mail subscription: | 1646 | 1529 |
| C. Total pd./requested circulation: | 1659 | 1539 |
| D. Free distribution by mail, carrier or other means: samples, complimentary and other free copies: | 95 | 140 |
| E. Total distribution: | 1754 | 1679 |
| F. Copies not distributed: | | |
| 1. Office use, left over, unaccounted, spoiled after printing: | 180 | 71 |
| 2. Return from news agent: | 0 | 0 |
| G. TOTAL: | 1934 | 1750 |

11) I certify that the statements made by me above are correct and complete
/s/ Anna Brereton, Circulation Manager

```
    Page# 2        COMB-MSS.BLK


     6
@ \  BUBBLE1
 1
 2 VARIABLE  -SWITCH
 3
 4 : BUBBLE1   ( -- )
 5     BEGIN  -SWITCH ON
 6         ITEMS 1- @  DO
 7             I 1+ S@  I S@ COMPARE 0(
 8             IF I  I 1+  EXCHANGE  -SWITCH OFF THEN
 9         LOOP
10     -SWITCH @  UNTIL ;
11
12
13
14
15


     7
@ \  COMB1
 1
 2 VARIABLE -SWITCH      VARIABLE  GAP
 3
 4 : COMB1  ( -- )
 5     ITEMS GAP !
 6     BEGIN  GAP @ 10 13 */ 1 MAX GAP !
 7     -SWITCH ON
 8     ITEMS GAP @ - 0
 9     DO
10         I GAP @ + S@  I S@  COMPARE  0(
11         IF I  I GAP @ +  EXCHANGE -SWITCH OFF  THEN
12     LOOP
13     -SWITCH @  GAP @ 1 =  AND UNTIL ;
14
15


     8
@ \  COMB2
 1
 2 VARIABLE -SWITCH      VARIABLE  GAP
 3
 4 : COMB2  ( -- )
 5     ITEMS GAP !
 6     BEGIN  GAP @ 10 13 */ 1 MAX
 7         DUP DUP 9 = SWAP 10 = OR IF DROP 11 THEN GAP !
 8     -SWITCH ON
 9     ITEMS GAP @ - 0 DO
10         I GAP @ + S@  I S@  COMPARE  0(
11         IF I  I GAP @ +  EXCHANGE -SWITCH OFF  THEN
12     LOOP
13     -SWITCH @  GAP @ 1 =  AND UNTIL ;
14
15
```

# FORML CONFERENCE

*The original technical conference*
*for professional Forth programmers, managers, vendors, and users.*

Following Thanksgiving, November 29–December 1, 1991

Asilomar Conference Center
Monterey Peninsula overlooking the Pacific Ocean
Pacific Grove, California U.S.A.

## Theme: Simulation and Robotics

Papers are invited that address relevant issues in the development and use of Forth in simulation and robotics. Virtual realities, robotics, and graphical user interfaces are topics of particular interest. Papers about other Forth topics are also welcome.

Attendees are invited to enter a robot in a robotics contest where the robot solves a puzzle.

Mail abstract(s) of approximately 100 words to **FORML Conference, Forth Interest Group, P.O. Box 8231, San Jose, CA 95155.**

Completed papers are due November 1, 1991.

## Conference Registration

Registration fee for conference attendees includes conference registration, coffee breaks, and note-book of papers submitted, and for everyone rooms Friday and Saturday, all meals including lunch Friday through lunch Sunday, wine and cheese parties Friday and Saturday nights, and use of Asilomar facilities.

Conference attendee in double room—$350 • Non-conference guest in same room—$200 • Children under 17 years old in same room—$140 • Infants under 2 years old in same room—free • Conference attendee in single room—$450
*Forth Interest Group members and their guests eligible for ten percent discount on registration fees.*

Register by calling the Forth Interest Group business office at (408) 277-0668 or writing to: **FORML Conference, Forth Interest Group, P.O. Box 8231, San Jose, CA 95155.**

*(Letters, from page 5.)*
one more reason that Forth is not more widely used. Isn't it time to start informing your readers about real Forth systems from real vendors committed to providing complete systems?

Sincerely,
Jim Callahan, President
Harvard Softworks
P.O. Box 69
Springboro, Ohio 45066

### IIe Solutions

Dear Editor,

In reply to Keith Brewster (*FD* XIII/2), since 1984, I have used muSpeed II, a special Forth for the Apple IIe. It consists of a processor card (Intel 8231A and arithmetic chip) and two diskettes (under DOS 3.3). Its characteristics: single- and double-precision math (16 and 32 bits). All floating-point operations are 32 bits. Range: 0,9223367 E+19. Also, it may use RAM expansion cards. The card-and-language system is a product of Applied Analytics, Inc. (8910 Brookridge Dr., Upper Marlboro, Maryland 20772). Also, you may use GoFORTH under ProDOS (IIe, IIgs) from Pair Software. Or MasterForth with floating point, from MicroMotion.

Today, I prefer F-PC running in an 80286-80287.

Sincerely,
Luis de la Cerda Delpin
Universidad de Chile
Casilla 13706
Santiago, Chile

### Singapore Slingshot Targets FIG Issues

Dear Editor,

With reference to the letter titled "Black-Belt Exhaustion & Lean, Mean FIG" (*FD* XIII/3), we are truly

surprised that FIG currently has only 2000 members. Does that include international members? We Forthians must have more than 2000 members in business using Forth in one way or another, so what went wrong?

The reason, we think, Harris abandoned its Forth efforts is obvious: the root is always money. If it is a hot product, we should be seeing the third generation of it by now.

Let me tell you the story of how our company got into Forth. It will explain my next suggestion on how to increase the membership figures and, more importantly, how to get more resources and attention from third-party vendors in order to make money.

My company specializes in making Eurocard, STD-bus-type controller boards and peripherals. Initially, we used assembly-language software monitors to run those boards. We found that customers had problems trying to debug such programs, especially when the equipment was pre-installed on site. So we looked around for a high-level, user-friendly, and interactive language that is also small, fast, and has almost all the advantages and convenience of a PLC (programmable logic controller).

We tried BASIC before coming to Forth. Since then, all our products have been programmed in Forth and assembler, and it has been used in a wide variety of applications, especially building and machine real-time automation. The interactive, real-time nature of Forth facilitates tuning on-the-fly like no other language.

In an effort to improve our programming skills and knowledge, we tried to buy all the Forth software and tools on the market. We began to realize that, slowly but surely, Forth tools and systems are being removed from vendors' product listings, or else the tools are outdated. We remember the times when most major magazines carried Forth articles and advertising.

Without self-sustaining third-party support (i.e., anyone using or promoting Forth must make money), Forth will become outdated due to too little economic activity.

We have some statements, experiences, and suggestions to share with you. Some of them may already have been thought of, and we apologize if we offend anyone by any of the suggestions or statements. We would like *FD* to comment on all the following.

The objective is to reestablish ourselves at least as a viable, ongoing, bankable language. (Note: some of these statements tend to become self-fulfilling, or chicken-and-egg problems; some of them may overlap.)

1. We found that through FD we learn a lot about the state-of-the-art in Forth, but nothing that will benefit the average (majority) user. Therefore, ordinary mortals (us) who normally buy computer magazines just for the Forth articles, would not buy *FD* or join FIG, because it is of no economic and immediate educational value.

2. We (especially companies) also buy computer magazines just to see what are the latest products, tools, and previews on the market, so why not *FD*?

3. We get a little shaky if Forth and its tools begin to become dated. I.e., who would want to produce state-of-the-art products for a market of a few thousands?

4. We have a very keen interest in hardware that can be used by Forth, semiconductors as well as board-level devices. We do not see any vendors given free space, as in *EDN* or *Electronic Design*, for application articles. We would buy the magazine just for such an article. Maybe even ask the vendor to pay a little for the promotion space.

5. Maybe *FD* is unable to do the above because it is a private magazine. Well, gentlemen, it is time to open up. Otherwise, the world will pass us by and it will be so sad, especially now that we already have the language on silicon and restricted marketing.

6. The day that one of the majority-supported languages acquires Forth's interactive characteristics, Forth will be dead.

7. Since Forth is good in real-time and control applications, include in every issue of *FD* one or more related articles (repeating every few years, if necessary, to ensure maximum coverage).

8. Anyone who makes money using Forth will have no problem buying one year's subscription to a Forth magazine, provided the magazine has some practical use (to everyone?) at all.

9. *FD* should use the example of major computer magazines, but with a difference. Use the characteristics of clannish and cultist Forthians to cultivate a readership.

10. Make *FD* into *The Forth Magazine*—attracting all people by carrying any


# UTIL

## A Forth Programming System For Palmtop Computers

Turn your palmtop computer into a portable software development system with UTIL, a small and fast implementation of Forth. Optimized for the Atari Portfolio and Hewlett-Packard 95LX, a PC version of UTIL is also available.

- Small kernel of just 8K and entire system of under 24K maximizes space for your data and programs
- Uses text source files created and modified with your palmtop editor
- Includes Forth compiler and decompiler, 8086 assembler and disassembler, comprehensive user's guide with examples and games
- Metacompiler and i860 development kit options
- Source code available for utility files

UTIL is priced from just $70. To order your copy, call today.

**Essex Marketing Services Inc.**
**272 Old Farms Road Simsbury CT 06070**
**Phone (203) 651-8284 Fax (203) 676-9481**

# SIGFORTH '92 CALL FOR PAPERS

In conjunction with the ACM Computer Science Conference, ACM SIGCSE Computer Science Education Conference and the ACM SIGAPP Symposium on Applied Computing

## FORTH LANGUAGE WORKSHOP

**March 5-7, 1992** **Kansas City, Missouri**

Come share recent work on the Forth language and its use in applications. Expose the CSC, SIGCSE and SIGAPP conference participants that register for our conference to Forth through the presented papers and tutorials. Discount cross-registration with CSC, SIGCSE and SIGAPP is available.

Papers on all aspects of Forth are invited. Here are some ideas:

| | |
|---|---|
| Forth in Education | Development Environments |
| ANS Forth Compliance | Software Management |
| Formal Methods | Forth Engines |
| Object Oriented Forth | Optimizing Compilers |

**Refereed Papers:** 100 word abstract by December 1, 1991 and a draft paper by January 1, 1992 (max. 15 pages). Advantages of submitting a refereed paper include feedback from other experts, possible presentation time during the CSC conference overlap track, other preferential presentation times and preferential proceedings space.

**Unrefereed Papers:** abstract by February 15, 1992 and a final paper by the conference.

Request an author's kit from the program chair for format information. All submissions will get presentation time and proceedings space.

### Features

Keynote Speaker: Charles Moore, Session on Software Management featuring Mike Wong of IBM, tutorial Introduction to Forth, Panel Session on "Moving from the Classroom to the Real World", ANS Forth Roundtable, and other invited speakers. A separate half day tutorial on ShBoom, a 100+Mhz stack-based 32-bit RISC microprocessor, and Open Boot: Portable Forth-based Firmware, will also be held.

---

**Hosted By:**

Digalog
Shaw Laboratories
Nanotronics Inc.

For paper submission information contact:

*Program Chair*
Dr. Paul Frenger
P.O. Box 820506
Houston, TX 77282-0506
(713) 589-9040
GEnie: P.FRENGER

To assist in conference organization or for special presentations contact:

*Conference Chair*
George Shaw
Shaw Laboratories Limited
PO Box 3471
Hayward, CA 94540-3471
(510) 276-5953, 276-6050 fax
GEnie: G.SHAW1
email:george_shaw@mts.cc.wayne.edu

*Program Committee*
Dr. Alan Furman Dr. Nicholas Solntseff
Martin Fraeman John Hayes
Irving Montanez Dr. Harvey Glass

*Sponsored by the ACM Special Interest Group on Forth*

and all types of articles, as long as they are related to Forth.

11. Before *FD* has the mass-market clout, encourage all types of advertising—Forth or otherwise—at cost or slightly higher. This will attract all vendors (pooling the marketplace), thus attracting users.

12. Use cheaper paper, if possible, because it is the content that will ultimately attract the money and, therefore, ensure survival as an entity to fight and propagate the Forth art. (No food, no art.)

13. Do anything possible to attract mass-market attention, even if we have to sacrifice some of the purists.

14. Allow vendors to write about their products and, if necessary, help them to present it on paper at cost.

15. Have a reader-service card, if possible.

16. Help vendors to port their products to Forth, and advertise this.

17. Have a service where hardware vendors can use *FD* as a trading house, just as *FD* is doing with software and books by mail order.

18. Start a vendor query column in which readers can question vendors. Vendors whose answers are published should pay.

19. Write a super-duper, compact version of Forth—first one for embedded systems, because it is easier, and later a version for disk-based systems. Include all the works, trappings, warts, and porting information, and give it to all vendors with the only condition that they can add to it but not change it. This will instantly establish a world-wide Forth standard. Do not worry

*(Continued on page 27.)*

# New Stack Tools

*Peter Verhoeff*
*Glendale, California*

Forth is a wonderful programming language. After all, what other language will let you add new commands by typing in their definitions, or execute algorithms by typing their names?

However, one thing I have personally had difficulty with is keeping track of what's on the stack and how to manipulate its contents. For example, in working with strings it is not uncommon to have six items—that is, three string addresses and three string lengths—on the stack. To keep all these in the right place can be quite a trick.

Since it was time for me to write another article for *Forth Dimensons*, I decided to tackle the subject of simplifying stack manipulation and share my findings with the readers. Perhaps some useful things would come to light, which might make a Forth programmer's life easier.

### Background

Back in the mid-eighties, I read something on that topic in *Forth Dimensions*. I believe the author of that article created stack words where, for example, to reverse six items on the stack, you would say something like

```
S" ABCDEF|FEDCBA"
```

The six letters to the left of the vertical bar would represent the starting stack picture and those to the right the result of the operation.

Since I no longer have the article, I am not sure about exactly how this was done, but I believe that the stack was first unloaded to a storage area, from which items would then be pushed back onto the stack in the desired sequence.

### First Approach

The first simplification I made to the above method was to replace the string to the left by a single letter. The above stack picture thus became:

```
S" F|FEDCBA"
```

where the first letter F, being the sixth letter of the alphabet, indicated that there were six items on the stack. Using a letter, rather than a numeral, would allow 26 stack items to be represented. Later, I created a separate word to dump the stack contents, which you would do once and then load from the storage area whenever you needed stack items.

I created an algorithm with which to do this and it performed very well. The code to do this was simple and word definitions using this type of stack notation looked a lot less cryptic than the usual definitions with the DUPs, ROTs, SWAPs, and so on. What's more, it was easy to figure out what was being done to the stack in a word definition, by looking at these new stack words.

A further refinement I put in was after I realized that stack items often would be incremented or decremented. I therefore wrote some code to recognize the four arithmetic operators +, -, *, and /, as well as the numerals 0 through 9 in the stack picture. Thus, to increment a string's address by 1 and reduce its length by 1, you would say,

```
S" A1+B1-"
```

(where A = address and B = length.)

### Second Approach

However, I was not quite happy yet. First, there was the fact that the stack was no longer used as a stack, since its contents were being dumped to a storage area, which was somewhat of a violation of its purpose. More important, however, was that the storage space for the stack data was being shared by each occurrence of this "stack string." This meant that any words between stack strings potentially would mess up the data in the storage area if their own definitions also contained stack strings.

When I realized this, I took a long, hard look at the purpose of the exercise. I discovered that there were really three different purposes:

a. Rearranging the stack contents, without regard to the mechanics of how this was done. This would be useful in testing and debugging of algorithms.

***Note that there are only three basic stack operations: ROLL, PICK, and DROP...***

b. Finding out the "stack primitives," such as DROP, ROT, SWAP, etc., that would yield the specified ending stack picture from a given starting stack picture. This would come in handy if you wanted to write new code in the conventional fashion.

c. Creating new stack words from existing stack words, for use in frequently occurring stack patterns.

All three objectives have been achieved in the following code. The task was harder than anticipated, but I believe it was worth the effort in creating a useful set of tools.

The Forth used for the code below was F83 Version 2.1. It is quite possible that there is a shorter and more elegant way of accomplishing the same result. Consider my efforts as a prototype.

**Stack String Examples**

Before delving into the code, let's take a look at some examples of stack strings. The simplest one is A|, which takes the top item off the stack and DROPs it. Likewise, B| represents 2DROP, since the B to the left of the verical bar (|) represents AB. B|A does the same thing as A|, but it assumes that there are two items on the stack. By the way, it does not matter how many items are actually on the stack, as long as there are at least as many items as represented by the letter to the left of the vertical bar. If a stack string starts with the letter F, you will need at least six items on the stack to execute it.

Instead of the vertical

```
Scr # 0         STACKS.BLK
 0 \  STACK MANIPULATION, USING STACK STRINGS.
 1
 2
 3    Copyright 1991, by Peter Verhoeff
 4
 5    P.O. Box 10424
 6
 7    Glendale, CA 91209
 8



Scr # 1         STACKS.BLK
 0 \  Load Screen.
 1
 2  2 16 THRU
 3



Scr # 2         STACKS.BLK
 0 \  Primitives and strings.
 1
 2 CREATE S$0 81 ALLOT          \ Text string.
 3 CREATE S$1 28 ALLOT          \ Starting string (pseudo stack)
 4 CREATE S$2 28 ALLOT          \ Ending   string (pseudo stack)
 5 VARIABLE .FLG  VARIABLE LTR  \ Display flag, letter variable.
 6
 7 : C+!  (S # adr -- )        \ Increment contents of adr by #.
 8    TUCK C@ + SWAP C! ;
 9
10 : $+$  (S adr len adr_to -- ) \ Append 1st string to 2nd one.
11   >R TUCK R@ COUNT + SWAP CMOVE  R> C+! ; \ Update count too.
12
13 : $    (S -- )                \ Enter a string from keyboard.
14    BL  PARSE-WORD  ;
15
```

## Advertisers Index

```
Scr # 3          STACKS.BLK
 0 \  General purpose words.
 1
 2 : UC? (S char -- t|f)          \ True if upper case.
 3    ASCII A  ASCII Z  BETWEEN ;
 4
 5 : LC? (S char -- t|f)          \ True if upper case.
 6    ASCII a  ASCII z  BETWEEN ;
 7
 8 : OP?   (S char -- t|f)        \ Check if arithmetic operator.
 9    DUP  ASCII + =    OVER ASCII - = OR
10    OVER ASCII * = OR SWAP ASCII / = OR ;
11
12 : NUM?  (S char -- t|f)        \ Check if numeric.
13    ASCII 0  ASCII 9  BETWEEN ;
14
15
```

```
Scr # 4          STACKS.BLK
 0 \  More general purpose words.
 1
 2 : LC  (S char -- char')        \ Convert char to lower case.
 3    DUP UC? IF BL + THEN ;
 4
 5 : -SCAN (S adr len char - adr' len') \ Reverse scan for char.
 6    -ROT TUCK + 1- SWAP 0 TUCK  \ Start at end of string.
 7    ?DO DROP 2DUP C@ =
 8    IF I 1+ LEAVE ELSE 1- THEN 0
 9    LOOP  ROT DROP ;
10
11
12
13
14
15
```

```
Scr # 5          STACKS.BLK
 0 \  Text string primitives.
 1
 2 : S$1+C (S char -- )           \ Append char to origin string.
 3    S$1 COUNT + C!  1 S$1 C+! ;
 4
 5 : S$0+$ (S adr len -- )        \ Append string to text string.
 6    S$0 $+$  1 S$0 C+!  ;       \ Put a trailing space.
 7
 8 : S$0+C (S char -- )           \ Append char to text string.
 9    S$0 COUNT + C!  2 S$0 C+! ; \ Put a trailing space.
10
11 : S$0+# (S # -- )              \ Append number to text string.
12    ASCII 0 OR  S$0+C  ;        \ Store as ascii numeral.
13
14 : S$0I  (S -- )                \ Initialize text string.
15    S$0 81 BLANK  ASCII : S$0 1+ C!  2 S$0 C! ;
```

bar, we can use the ampersand (&), which indicates that the starting string is to be repeated in the ending string. Thus, D&AB is the equivalent of D | ABCDAB, or 2OVER.

Not only is this a shorter way of writing the same thing, it allows you to switch between saving or not saving the original pattern by changing a single character. For example, D | AB DROPs items C and D from the stack and retains A and B (i.e., 2DROP), whereas D&AB PICKs items A and B, leaving the original ABCD intact at the bottom (i.e., 2OVER.)

### How to Use Stack Strings

I created two different high-level stack-string interpreters, as shown in screen 14. The first one, SM, is for use within a colon definition. SM does not create a new stack word, but instead executes the stack string. An example of this is given in screen 15. This screen will be discussed later, but the thing to observe here is the format, which is:

```
" <stack string>" SM
```

The SD command (screen 14) takes a stack string and displays its definition. For example, to see the definition for D&AB, you type:

```
$ D&AB SD
```

The displayed result will be:

```
: D&AB 2OVER ;
```

If you wish, you can then compile this word by typing:

```
SC
```

The $ command (screen

two) takes the typed-in text string which follows and puts its address and length on the stack. If you want to execute a stack string from the keyboard, without analyzing or compiling it, you can instead type:

```
$ D&AB SM
```

### Code Example

Screen 15 has an actual example of a word definition using stack strings. It looks for multiple occurrences of substring $1 within text string S$0 and replaces them with the shorter substring $2. To do this, we have to keep at least six items on the stack. Note the consistent use of the ampersand in the stack strings to retain the six bottom items, except for the F| at the end, which drops the six items. The first letter is F in all but two stack strings, where a seventh item is added and thus becomes G. The stack strings F&BD- and F&0CED could have been written as E&AC- and D&0ACB with the same results, but sticking with the letter F makes it easier to understand, since this way each letter has a consistent meaning.

The actual use for SR is to simplify the contents of the text string (see screen 16). Since the primitives in the text string are machine generated, without taking all the rules into account, there are certain simplifications that can be carried out. For example, SWAP SWAP can safely be omitted, since the stack contents would be the same as before the SWAPs. This algorithm is only used in the text string, since in direct execution this substitution is unnecessary and will execute correctly

```
Scr # 6          STACKS.BLK
 0 \  Letter-to-number conversion and logic for operators.
 1
 2 : SL># (S ltr -- adr len )   \ Find offset for letter.
 3    S$1 COUNT 3DUP + C!       \ Save the letter, same count.
 4    ROT -SCAN  ;              \ See if letter occurs in string.
 5
 6 : S2>1  (S -- )              \ 2 items replaced by 1 result.
 7    -2 S$1 C+!                \ Reduce item count.
 8    1 LTR C+!  LTR @ S$1+C  ; \ Use next available letter.
 9
10
11
12
13
14
15

Scr # 7          STACKS.BLK
 0 \  Roll instruction determined by letter.
 1
 2 : SROLL (S ltr -- )                \ Stack roll per letter.
 3    UPC  DUP   SL>#  DUP            \ Check if valid.
 4    IF TUCK OVER 1+ -ROT 1+ CMOVE   \ Update pseudo stack S$1.
 5       NIP 1-  ?DUP
 6       IF .FLG @                    \ 0 roll = do nothing
 7          IF DUP 1 =
 8             IF DROP " SWAP"        \ 1 roll = swap
 9             ELSE DUP 2 =
10                IF   DROP " ROT"    \ 2 roll = rot
11                ELSE S$0+# " ROLL" \ Standard roll
12             THEN THEN S$0+$        \ Put into display string
13          ELSE ROLL                \ Execute if display flag off
14       THEN THEN
15    ELSE 2DROP CR EMIT ."  invalid" THEN ;

Scr # 8          STACKS.BLK
  0 \  Pseudo stack pick and drop.
 1 : SDROP (S ltr -- )                \ Roll per letter and drop.
 2    SROLL .FLG @ IF " DROP" S$0+$ ELSE DROP THEN -1 S$1 C+! ;
 3
 4 : SPICK (S ltr -- )                \ Pick from stack per letter.
 5    DUP SL>#  DUP                   \ Check if valid.
 6    IF NIP 1- .FLG @
 7       IF  DUP 0=
 8          IF DROP " DUP"            \ 0 pick = dup
 9          ELSE DUP 1 =
10             IF   DROP " OVER"      \ 1 pick = over
11             ELSE S$0+# " PICK"     \ Standard pick
12          THEN THEN S$0+$           \ Put into display string
13       ELSE NIP PICK THEN           \ Execute if display flag off
14       S$1 C@ 1+ S$1 C!             \ Update the count.
15    ELSE 2DROP CR EMIT ."  invalid" THEN ;
```

```
Scr # 9          STACKS.BLK
 0 \  Pseudo stack initialization.
 1
 2 : SI5  (S adr len -- adr'len') \ Return effective S$2 string.
 3    S$2 COUNT 2SWAP 0 OVER      \ Setup.
 4    IF DROP SWAP COUNT TUCK LC? \ Check if 1st char lower case.
 5       IF 1 3 ROLL 0            \ If so, check rest of string.
 6          ?DO DROP COUNT TUCK  3 ROLL - 1 <>
 7             IF I 1+ LEAVE THEN 1
 8          LOOP                 \ Loop while next letter is next
 9       ELSE DROP 0 THEN
10    THEN -ROT 2DROP /STRING ;   \ Skip those letters.
11
12
13
14
15



Scr # 10          STACKS.BLK
 0 \  Pseudo stack initialization.
 1 : SI3  (S -- )                 \ Normalize pseudo stacks.
 2    S$1 COUNT TUCK + SWAP 0     \ Set up.
 3    ?DO 1- >R R@ C@ S$2 COUNT
 4       2 PICK 3DUP LC SCAN NIP  \ Check if S$1 char lc in S$2.
 5       IF DROP 2DROP  ELSE SCAN \ Check if S$1 char uc in S$2.
 6       IF SWAP LC SWAP C!       \ If uc, make lc.
 7       ELSE DROP SDROP THEN THEN R> \ Drop unused items.
 8    LOOP DROP  ;
 9
10 : SI4  (S -- adr len )         \ Return effective S$2 string.
11    S$2 COUNT OVER 0 2SWAP 0
12    ?DO COUNT OP?
13        IF LEAVE
14        THEN SWAP 1+ SWAP
15    LOOP  DROP 2- 0 MAX ;       \ If operator, back up two.



Scr # 11          STACKS.BLK
 0 \  Pseudo stack initialization.
 1
 2 : SI1 (S adr len -- adr'len')  \ Process origin pseudo stack.
 3    OVER C@ DUP LTR ! ASCII @ XOR DUP \ Get # of stack items.
 4    S$1 C! ASCII A S$1 1+ ROT 0 \ Set up 1st string.
 5    ?DO 2DUP C! 1+ SWAP 1+ SWAP \ Store letters in 1st string.
 6    LOOP 2DROP 1 /STRING ;      \ Prepare for 2nd string.
 7
 8 : SI2 (S adr len -- )          \ Do destination pseudo stack.
 9    S$2 OFF OVER C@  ASCII & =
10    IF S$1 COUNT S$2 PLACE      \ Copy S$1 if separator = '&'.
11   THEN 1 /STRING S$2  $+$  ;  \ Append rest of original string.
12
13 : SI (S adr len -- adr'len')   \ Prepare pseudo stacks.
14    SI1  SI2  SI3  SI4  SI5 ;
15
```

without it. With extensive use of stack strings, more simplifications may come to light, which may then be added to the SN' definition.

Perhaps a better method would have been to use tokens instead of string substitutions, but I decided against that, since it would have required considerable rewriting of the code, without altering the basics. Please note the leading and trailing space in both strings. The SN' algorithm is fairly slow in execution, but I believe that this can be improved considerably by searching for upper case only, or other optimization routines. Of course, compiling the stack strings in screen 15, or replacing them with the equivalent primitives would speed things up too.

### Pseudo Stacks

Let's take another look at screen 14. SM, since it executes the stack string directly, checks whether there are enough items on the stack and aborts if this is not the case. SD does not have that requirement, since it only creates the definition.

Three internal strings are used (see screen two). First, there is the text string S$0, used by SD, to build a colon definition of the stack string. Next, there is starting string S$1, which contains a representation of the current stack picture. Third, there is the ending string S$2, which contains a representation of the stack configuration we want to end up with. I have named these two strings "pseudo stacks," since they reflect what goes on on the stack.

### Initialization

Both SD and SM initial-

ize the pseudo stacks. This is a fairly complex process (see screens nine through 11.) SI, on screen 11, is the overall initialization word, which contains five components, SI1 through SI5. SI1 prepares starting string S$1; it creates a string of consecutive letters from the first letter of the input string. For example, if that letter is F, it will put ABCDEF in S$1. Next, SI2 first checks the separation character ( | or & ). If it is an &, it places a copy of S$1 into S$2. Any other character here is ignored. Next, the balance of the input string is appended.

SI3 is a little more complex. Let's use an example to illustrate its operation. Let's say that our input string is F&AB. After SI1 and SI2 have executed, S$1 will contain ABCDEF and S$2 will contain ABCDEFAB. Later on, in the main execution part of SD or SM, we will scan S$2 from left to right and put each item in turn on the top of the stack. Let's go through that process here.

| S$1 | Letter | Command |
|---|---|---|
| ABCDEF | A | 5 ROLL |
| BCDEFA | B | 5 ROLL |
| CDEFAB | C | 5 ROLL |
| DEFABC | D | 5 ROLL |
| EFABCD | E | 5 ROLL |
| FABCDE | F | 5 ROLL |
| ABCDEF | A | 5 PICK |
| ABCDEFA | B | 5 PICK |
| ABCDEFAB | | |

You can see that each operation is either a ROLL or a PICK. The basic rule is that a ROLL is executed the first time an item is encountered in S$2. Any subsequent occurrence of that letter in S$2 will become a PICK. If it were a ROLL also, the first occurrence would be wiped out. To differentiate between ROLLs and PICKs, we check each of the characters in S$1. If the character is found in S$2, we change the first occurrence to lower case. In our above example, therefore, the ABCDEFAB in S$2 will be converted to abcdefAB. Later, when we process S$2, we will do a ROLL when we encounter a lower-case character and a PICK when we find an upper-case character.

Thus we get:

| S$1 | Letter | Command |
|---|---|---|
| ABCDEF | a | 5 ROLL |
| BCDEFA | b | 5 ROLL |
| CDEFAB | c | 5 ROLL |
| DEFABC | d | 5 ROLL |
| EFABCD | e | 5 ROLL |
| FABCDE | f | 5 ROLL |
| ABCDEF | A | 5 PICK |
| ABCDEFA | B | 5 PICK |
| ABCDEFAB | | |

The above method is valid, but you have probably noticed that putting 5 ROLL six times was unnecessary and that all you really needed to do was 5 PICK twice. Here we come to the second rule, which states that if the first character in S$2 is lower case, that character and any lower-case letter that follows it—if it is the next letter in the alphabet—is to be ignored in the processing. This is accomplished in intialization routine SI5. Thus we get:

| S$1 | Letter | Command |
|---|---|---|
| ABCDEF | A | 5 PICK |
| ABCDEFA | B | 5 PICK |
| ABCDEFAB | | |

We are not done with SI3 yet. Let's consider D | AC. This would translate to ABCD in S$1 and ac in S$2. Notice that B and D do not occur in S$2 and are therefore not needed. This brings about the third rule: Any stack item in S$1 which does not occur in S$2 is dropped before S$2 is processed. We would do ROT 2DROP to execute D | AC.

Before we get done with SI3, let's take a look at SI4. Let's say we want to type part of a string whose address and length are on the stack, but ignoring the first n characters (stack picture: adr len n). For this, you would create the stack string C | AC+BC-, which would create ABC in S$1 and ac+bC- in S$2. Let's analyze what would happen:

| S$1 | Letter | Command | |
|---|---|---|---|
| ABC | a | 2 ROLL | ( ROT ) |
| BCA | c | 1 ROLL | ( SWAP ) |
| BAC | + | + | ( D = A + C ) |
| BD | b | 1 ROLL | ( SWAP ) |
| DB | C | Error! | C no longer exists! |

We should have done 1 PICK (OVER) instead of 1 ROLL when we encountered the letter c. That way, we would still have a C to use later on. Rather than writing some complex logic to handle this, I decided on a different approach, which is to allow the user to decide which occurrence to ROLL by making it lower case. For example, if you use C | AC+BC-, you will get an error; but if you enter C | AC+Bc- (last c in lower case), you won't get the error. SI3 first checks if the letter from S$1 occurs in lower case in S$2. If it does, no further checking is done on that character. Of course, if you compile a word like that, all the letters will be made upper case (at least in the version of Forth I have), but at that point it no longer matters, since the word has already been defined.

Also notice that I ignored rule number two (the one that says to skip lower-case letters at the beginning of the string). This is because it doesn't fully apply in the case of an arithmetic operation: you have to have both items at the top of the stack to carry out the operation. SI4 checks for this. It looks for an arithmetic operation and, if found, it goes back two places and ends the string right there, at least as far as SI5 is concerned. Any characters before that point are inspected by SI5, but nothing beyond that point. When SI5 is done, it returns the entire S$2 string, minus any leading characters at the start of the string that can be ignored.

**String Processing**

Let's take a look at SD' on screen 13. It processes the modified S$2 string, passed on by SI5. It inspects each character, from left to right, and determines whether it is a lower-case letter, upper-case letter, numeral, arithmetic operator, or other character. The .FLG variable is set on to indicate that the results are to be displayed.

If the character is lower case, SROLL (see screen seven) is executed. SROLL

```
Scr # 12          STACKS.BLK
 0 \  Manipulate per pseudo stacks.
 1
 2 : SM'  (S itms adr len -- itms')    \ Execute stack string 2.
 3    .FLG OFF  0
 4    ?DO COUNT SWAP >R       DUP LC?
 5       IF SROLL        ELSE DUP UC?
 6       IF SPICK        ELSE DUP ASCII + =
 7       IF DROP + S2>1 ELSE DUP ASCII - =
 8       IF DROP - S2>1 ELSE DUP ASCII * =
 9       IF DROP * S2>1 ELSE DUP ASCII / =
10       IF DROP / S2>1 ELSE DUP NUM?
11       IF DUP S$1+C  ASCII 0 XOR
12       ELSE DUP S$1+C
13       THEN THEN THEN THEN THEN THEN THEN R>
14    LOOP  DROP  ;
15
```

```
Scr # 13          STACKS.BLK
 0 \  Manipulate per pseudo stacks.
 1
 2 : SD'  (S adr len -- ) .FLG ON  0     \ Interpret string S$2.
 3    ?DO COUNT SWAP >R
 4       DUP LC?  IF SROLL ELSE
 5       DUP UC?  IF SPICK ELSE            \ Roll
 6       DUP NUM? IF DUP S$0+C S$1+C ELSE \ Pick
 7       DUP OP?  IF S$0+C S2>1      ELSE \ Numeral
 8       DUP " ASCII" S$0+$ S$0+C S$1+C   \ Operator
 9       THEN THEN THEN THEN  R>          \ Other character
10    LOOP  DROP  ;
11
12
13
14
15
```

```
Scr # 14          STACKS.BLK
 0 \ High level stack manipulation words.
 1
 2 : SM (S itms adr len -- itms') \ Manipulate stack per string.
 3   OVER C@ ASCII @ XOR DUP 4 + DEPTH > \ Check the stack depth.
 4    IF CR  . ABORT" stack items needed" \ Stack underflow.
 5    ELSE DROP SI SM' THEN ;     \ Initialize, then execute.
 6
 7 DEFER SN
 8
 9 : SD (S adr len -- )          \ Define new stack word in S$0.
10    S$0I 2DUP  S$0+$  SI  SD' \ Build definition.
11    59  S$0+C  SN            \ Append semicolon, normalize.
12    CR  S$0 COUNT TYPE  ;     \ Display it.
13
14 : SC (S -- ) S$0 COUNT       \ Compile text string S$0.
15    TUCK TIB SWAP CMOVE  #TIB !  BLK OFF  >IN OFF  INTERPRET ;
```

first converts the character on the stack back to upper case, then executes SL># (screen six), which scans S$1 in reverse direction to locate an occurrence of that letter. It also appends a copy of the letter to the end of S$1, but without incrementing the character count.

If the letter is not found in S$1, an "invalid" message is displayed to indicate that the operation failed. If the letter is found, SL># returns its position on the stack (relative to the top) where the item occurs and S$1 is rearranged to move the letter from where it occurred to the end of the string and the number itself is decremented by one.

Next, we check the decremented number. If it is zero, no action is necessary, since 0 ROLL is in fact a no-operation. Otherwise, if .FLG (display flag) is false, a ROLL is executed to move the item to the top of the actual stack. If .FLG is true, we check the number further. If it is a one, we drop the number and move "SWAP" to the text string, since 1 ROLL is equivalent to SWAP. If it is a two, we also drop it and move "ROT" to the text string. If it is any other number, we convert it to ASCII, append it to the text string, followed by a space and the literal "ROLL".

Note that this special-case processing could also have been done in the top level SN' routine, but that routine had not been written at that point.

SPICK (screen eight) works similarly to SROLL. Special cases are 0 PICK (DUP) and 1 PICK (OVER). Also note that the character count of S$1 is incremented to account for the increased

stack depth.

SDROP, depending on .FLG, either executes a DROP or appends "DROP" to the text string, and in both cases decrements the count of S$1 to reflect the decreased stack depth.

Note that there are only three basic stack operations: ROLL, PICK, and DROP. All others can be broken down into permutations of those three.

Let's go back to SD' on screen 13. If the character under consideration is numeric, it is appended to the text string and to the pseudo stack S$1. If the character is an arithmetic operator, it is appended to the text string and S2>1 is executed. S2>1 (screen six) gets the next letter after the last one that was used, decrements the S$1 count by two, and then appends the new letter to S$1 (incrementing the count by one in the process.) The new letter is used to indicate that the result of the operation is a new value. Interestingly, this letter can be reused later on in S$2 and can be ROLLed, PICKed, or DROPped.

If the character being processed in SD' is neither a letter, a numeral, or an operator, it is appended to the text string as an ASCII character. This allows for a little extra flexibility in the use of stack strings, although I personally haven't found a use for it yet.

SM' is, of course, used by SM. It has a similar pattern to SD', but is used to execute, rather than work with, the text string.

**Summary**

This code works and should be a useful addition to the Forth programmer's tool set. A lot more work can be done on the subject of stack manipulation, and I welcome any further suggestions and feedback you may have.

```
Scr # 15          STACKS.BLK
 0 \  Substring substitution in S$0.
 1
 2 : SR  (S a1 l1 a2 l2 -- )  \ Replace all $1 with $2 in S$0.
 3    S$0 COUNT
 4    BEGIN " F&ABEF"     SM  SEARCH  \ Search for $1 in S$0.
 5      IF /STRING                    \ Start where $1 is found.
 6         " F&BD-"      SM           \ Compute l1 - l2.
 7         " G&EG+EFG-" SM  CMOVE     \ Move trailing part of S$0.
 8         " G&EF+G-G"  SM  BLANK -   \ Blank end, fix length.
 9         " F&0CED"     SM  CMOVE    \ Replace with $2, loop.
10      ELSE DROP TRUE   THEN         \ Exit if $1 not found.
11    UNTIL " F|"         SM          \ Clear the stack.
12    S$0 COUNT -TRAILING SWAP 1- C! ; \ Update length of S$0.
13
14
15

Scr # 16          STACKS.BLK
 0 \  Normalization of stack commands
 1
 2 : SN' (S -- )                     \ Normalize text string.
 3    "  SWAP SWAP "      "  "        SR
 4    "  ROT ROT "        "  -ROT "   SR
 5    "  1 + "            "  1+ "     SR
 6    "  2 + "            "  2+ "     SR
 7    "  2 * "            "  2* "     SR
 8    "  2 / "            "  2* "     SR
 9    "  SWAP + "         "  + "      SR
10    "  SWAP * "         "  * "      SR
11    "  OVER OVER "      "  2DUP "   SR
12    "  3 ROLL 3 ROLL " "  2SWAP " SR
13    "  3 PICK 3 PICK " "  2OVER " SR
14    "  DROP DROP "      "  2DROP " SR ;
15 ' SN' IS SN
```

## Articles Needed

*Forth Dimensions* depends on its readers—people just like you—to write about their versions of Forth utilities, interesting applications of Forth, a recent brainstorm, a new way of looking at an old problem, and issues about working in the real-life Forth world. Or write a tutorial, your ideas to make Forth and FIG more viable, or a letter that responds to a recent *FD* author.

Write to: Editor, *Forth Dimensions*, P.O. Box 8231, San Jose, California 95155

# QuikFind String Search

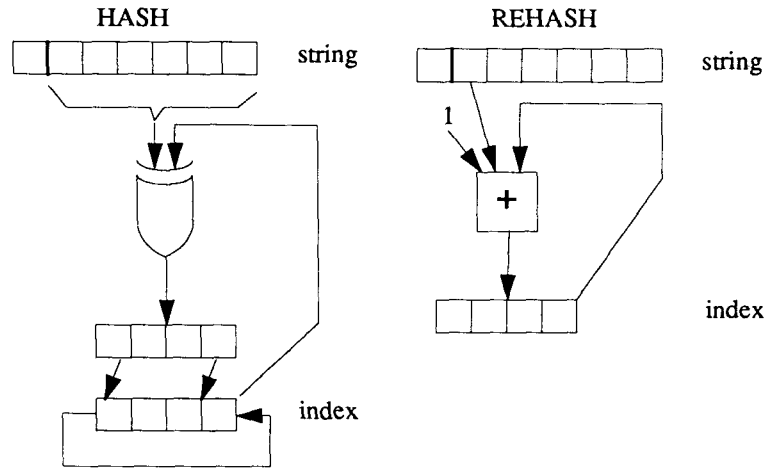
*Rob Chapman*
*Edmonton, Alberta*

This all started a few years ago compiling code on a 32-bit fig-FORTH.

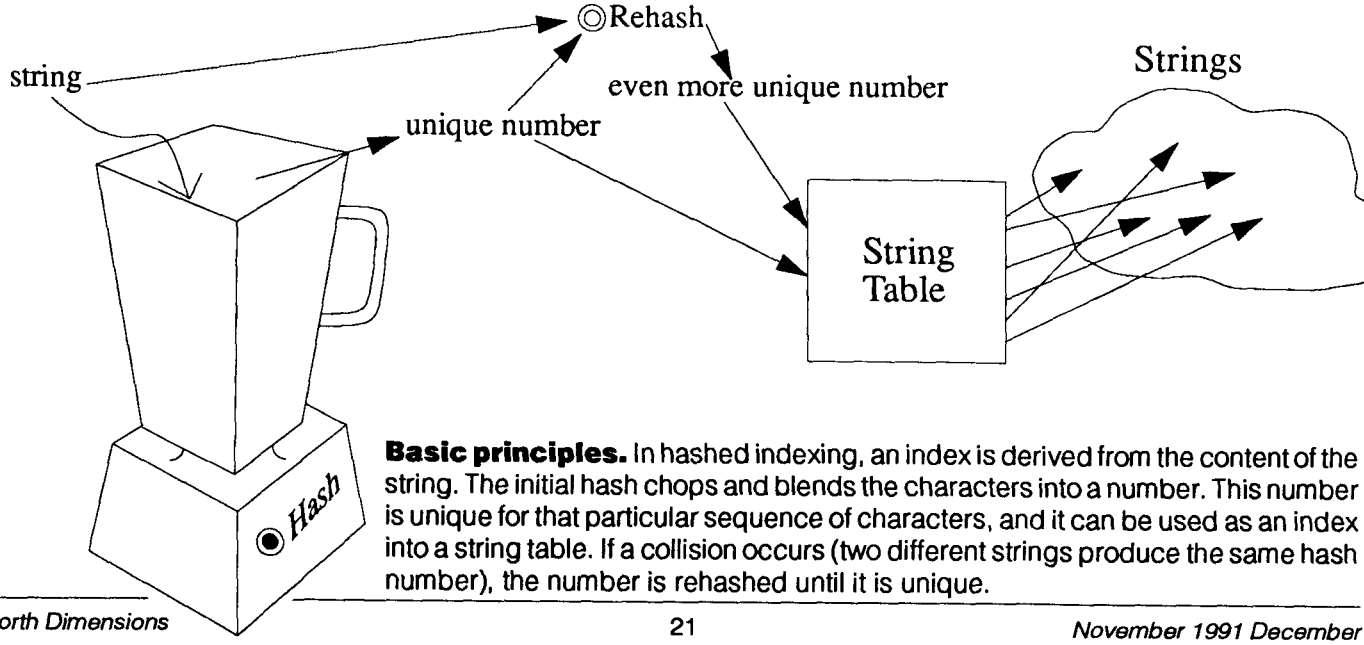
The dictionary contained thousands of words, and compilation often had to be started from the first file. This took a lot of time. I used this time to explore alternate dictionary look-up algorithms. Someone suggested a binary search and, since it is a fairly straightforward concept, I went ahead and implemented it. It greatly reduced compile times, and I was hooked on improving it further. Most people would say, "Oh yeah, but hashing would be faster," but they knew little beyond that. Since I didn't take Computing Science, I immersed myself in a course of self-study on hashing. I picked up bits and pieces from some Forth papers and a few textbooks. And then the fun began; I evolved my ideas through Forth.

I tried several schemes of turning strings

*(Continued on next page.)*




**Hash Algorithms.** The hash algorithm XORs each character of the string with the index. After each XOR, the bits in e index are rotated left three bits. The rehash algorithm adds the first character plus one to the index.




**Basic principles.** In hashed indexing, an index is derived from the content of the string. The initial hash chops and blends the characters into a number. This number is unique for that particular sequence of characters, and it can be used as an index into a string table. If a collision occurs (two different strings produce the same hash number), the number is rehashed until it is unique.

into numbers until I hit upon one which gave the greatest amount of unique numbers. Now that I had a good hash algorithm, I needed to find the optimal seed. I did this by letting the algorithm run continuously overnight, trying out every 16-bit number. It ran for about 20 hours, but one magic number stood out and I integrated it into the hashing algorithm.

With the primary hashing algorithm settled upon, I needed an equally good secondary hashing algorithm to handle collisions. Since it might be used several times, it would have to be simple and efficient. I settled upon using the first letter of the string to create secondary hashes.

While I developed the algorithm, I worked on the language as well, using many different factorings and names.

Once I reached a break-even point of diminishing returns (i.e., compile time was so short that I had no more time to reduce compile time), I stopped and thought about it for a couple of years.

It seemed like something was missing. The deficiencies in the algorithm bothered me. After a few gatherings with fellow FUGgers,[1] the deficiencies were characterized and some complicated solutions were available. Since they were complicated, there seemed to be too large a payoff.

Finally, the collection of everything in the grey matter spawned a new idea which simply addressed a major deficiency. I implemented it and it worked. Andrew Scott then fine-tuned it in a few places and incorporated it into the

**A Code Walk-Through**

This is QuikFind expressed in botForth. Each section of code is preceded with a discussion.

The number of locations in the string table should be a prime number to maximize the number of locations available for rehashing.

```
( QuikFind: fast string finder )
(  Rob Chapman  Oct. 24, 1990 )
HEX
  3FD CONSTANT #entries
  ( 1021 locations; should be a prime number )
  DATA string-table  #entries CELLS ALLOT
```

BLEND and ASCKEY are the algorithms of HASH and REHASH, respectively. BLEND pulls the next character out of the character string and exclusive-ORs it with the hash index. It then rotates all the bits to the left with the most-significant bit becoming the least-significant bit. This is done for each character in the string to obtain the primary hash index. The second algorithm, ASCKEY, uses the first character of the string to hop to the next location in the string table. One is added to the ASCII value of the character to prevent the null character from causing an endless loop.

```
( ==== 16-bit rotate left ==== )
: ROL  ( n -- n' )  DUP 2*
  SWAP 8000 AND  IF  1 OR  ENDIF ;

( ==== Hashing algorithms ==== )
: BLEND ( string \ n -- string' \ n' )
  >R C@+ SWAP  R> XOR  ROL ROL ROL ;

: ASCKEY  ( string \ loc -- string \ loc' )
  OVER 1 + C@ 1 + CELLS + ;
```

Once a location in the string table is hashed to, the strings are compared by MATCH?. CHARS pulls characters out of the two strings and MATCH? compares them. On the first byte, only the lower six bits are compared. The lower five bits are the count, and the sixth bit is a smudge bit. If the smudge bit is set, the strings won't match. Two strings of zero length will produce a match.

```
( ==== Short string compare;
  first byte: xx | smudge bit | 5 bit count ==== )
: CHARS  ( a \ a -- a+ \ a+ \ c \ c )
  COUNT >R >R  COUNT  R> SWAP  R> ;

: MATCH?  ( string \ name -- flag )
  CHARS  OVER XOR  3F AND  >R
  1F AND  R>
  BEGIN  0=  WHILE
    ?DUP  IF  1 - >R  CHARS XOR  R> SWAP
    ELSE  2DROP  YES  EXIT  ENDIF
  REPEAT  DROP  2DROP  NO ;
```

USED? and DIFFERENT? are used to interrogate a location in the string table. USED? returns a true if the location has something in it other than a zero or "0string" (0string is used to replace strings which have been removed from the hash table). DIFFERENT? compares the given string against the one pointed to. A true results if the strings match or if the location is zero.

```
( ==== Table checks ==== )
DATA 0string  0 ,
( zero-length-null-string for replacing deleted entries )

: USED?  ( loc -- f )
  @ DUP  IF  0STRING XOR  ENDIF ;

: DIFFERENT?  ( string \ loc -- f )
  @ DUP
    IF MATCH?  0=
    ELSE NIP ENDIF ;
```

HASH, REHASH, BUMP, and LOCATE can be considered as internal messages to the string table. HASH starts with the magic number D177 and blends all the characters into it. This number is MODed with the size of the hash table to obtain the location index. REHASH finds the next location, based on the ASCKEY algorithm. BUMP is used by INSERT to bump an older definition in the table, which happens when a word is redefined. LOCATE finds a given string in the table or the first zero location.

```
( ==== Messages for implicit string table ==== )
( internal: )
: HASH  ( string -- loc )
  D177  ( magic seed ) SWAP
  COUNT 1F AND
  FOR BLEND NEXT  NIP
  #entries MOD
  CELLS  string-table + ;

: REHASH  ( string \ loc -- string \ loc' )
  ASCKEY
  DUP  string-table
  #entries CELLS +  >
    IF  #entries CELLS -  ENDIF ;

: BUMP  ( string \ loc -- string' \ loc )
  DUP >R  DUP @ >R  !  R> R> ;

: LOCATE  ( string -- loc )
  DUP HASH
  BEGIN  2DUP DIFFERENT?
  WHILE REHASH  REPEAT  NIP ;
```

INSERT, APPEND, DELETE, QUIKFIND, and EMPTY can be considered as external messages to the string table. INSERT is used to insert a string into the string table. If it encounters a twin (i.e., a redefinition), then the string is inserted at that location and the twin is inserted after it. APPEND is used to insert a string into the table, as well. It differs in the fact that it does not bump definitions. INSERT and APPEND apply link-list functionality to the string table. When a string is rehashed, it is like moving to the next link in a link list. DELETE removes a string from the table. To maintain rehash lists, it must be replaced with another string. If it was replaced with a zero, it would be like truncating a link list (or several). 0string is a string that will never occur normally, so it is used as the hole filler. It may be replaced with another string. QUIKFIND accepts a string and finds a match or zero within the table. EMPTY is used to initialize the string table to all zeroes.

```
( external: )
: INSERT  ( string -- )  DUP HASH
  BEGIN  DUP USED?
```

compiler.

### The Final Piece

The major deficiency I refer to is the ability to delete single strings from the string table. Since this table replaces the traditional link list, it should provide equivalent functionality (insert, append, and delete). In the link list model of a Forth dictionary, it is easy to add some new words and then remove them (vocabulary or module scoping). In my first attempt at the hash algorithm two years ago, I simply reinstalled all the words into the table when any words were removed from the Forth link list. This took about half a second and discouraged the use of modules for scoping. It worked, but I wasn't happy.

The solution to this major deficiency was to replace the string to be removed with an empty string. Otherwise, if a string was deleted from the hash table and replaced with a zero, it might truncate other hash paths which bounce through this location. It's a simple and obvious idea, once you discover it. This empty string is a predefined string with a zero count byte.

This means there are three types of locations in the table: unused, used, and dirty. The unused locations contain a zero and indicate the end of the current rehash search path. The used location contains a pointer to the word name. The dirty locations point to the empty string. When searching for a string, the empty locations are skipped over. When searching for a place to insert a string (or append), the dirty locations may be used as well as the unused locations.

### A Few Measurements

When I did some comparisons between searching for words using a link list or the hash algorithm, the hash algorithm was anywhere from three to 450 times faster. The dictionary had about 250 words. The link list algorithm searched 125 words on the average, while the QuikFind algorithm searched about 1.2 words on the average. Although this is a major function of compiling, the compile times won't be decreased by such massive amounts, since there are other processes involved.

### Other Thoughts

The code included in this paper allows only one string table to be defined. This is sufficient for most needs, but if the ability to create multiple string tables were added, the QuikFind algorithm would be available for other uses. In this case, the table could be thought of as an object which received the messages INSERT, APPEND, DELETE, QUIKFIND, and EMPTY.

Rob Chapman is a software engineer at IDACOM, a division of Hewlett-Packard. He is currently on a mission to port the simplest Forth (botForth) to every platform (in the simplest way, of course).

1. Forth Users Group: weekly noon-hour rap sessions with Forth as a central topic.

```
    WHILE 2DUP @ MATCH?
      IF BUMP  ENDIF
  REHASH REPEAT  ! ;

: APPEND  ( string -- )
  DUP HASH
    BEGIN DUP USED?
    WHILE REHASH REPEAT  ! ;

: DELETE  ( string -- )
  LOCATE  0string SWAP ! ;

: QUICKFIND  ( string -- entry | 0 )
  LOCATE @ ;

: EMPTY  ( -- )
  string-table
  #entries CELLS 0  FILL ;
```

Here are two examples of how to hook QuikFind into the botForth compiler. INSTALL runs through the dictionary and installs all the words into the table. The redefinition of : creates a definition, unsmudges it, inserts it into the string table, and then smudges it. RECURSIVE unsmudges a word. If a word was smudged and inserted, it would not bump any previous definitions of the same name.

```
  ( ==== Sample application ==== )
: INSTALL  ( -- )
  EMPTY  LATEST
  BEGIN  ?DUP
    WHILE  @+ APPEND
  REPEAT ;

: :  ( -- )
  \ :  \ RECURSIVE
  LATEST INSERT  SMUDGE ;
```

### fig-FORTH to botForth

These are a few definitions which should allow the QuikFind code to run on fig-FORTH.

```
: CELL  ( -- n )  2 ;
: CELLS  ( n -- m )  CELL * ;

: YES  ( -- f )  -1 ;
: NO ( -- f )  0 ;

: NIP  ( n \ m -- m )  SWAP DROP ;

: C@+  ( a -- c \ a+ )  DUP C@  SWAP 1 + ;

: \  ( -- )  [COMPILE] [COMPILE] ; IMMEDIATE

: ENDIF  ( sys -- )  0 \ LITERAL  \ DO ; IMMEDIATE
: NEXT  ( sys -- )  \ LOOP ; IMMEDIATE

: DATA  ( -- )  0 VARIABLE  CELL NEGATE  ALLOT ;
```

# HARVARD SOFTWORKS

*NUMBER ONE IN FORTH INNOVATION*

(513) 748-0390 P.O. Box 69, Springboro, OH 45066

**MEET THAT DEADLINE ! ! !**

- Use subroutine libraries written for other languages! More efficiently!
- Combine raw power of extensible languages with convenience of carefully implemented functions!
- Yes, it is faster than optimized C!
- Compile 40,000 lines per minute!
- Stay totally interactive, even while compiling!
- Program at any level of abstraction from machine code thru application specific language with equal ease and efficiency!
- Alter routines without recompiling!
- Use source code for 2500 functions!
- Use data structures, control structures, and interface protocols from any other language!
- Implement borrowed feature, often more efficiently than in the source!
- Use an architecture that supports small programs or full megabyte ones with a single version!
- Forget chaotic syntax requirements!
- Outperform good programmers stuck using conventional languages! (But only until they also switch.)

**HS/FORTH with FOOPS - The only full multiple inheritance interactive object oriented language under MSDOS!**

Seeing is believing, OOL's really are incredible at simplifying important parts of any significant program. So naturally the theoreticians drive the idea into the ground trying to bend all tasks to their noble mold. Add on OOL's provide a better solution, but only Forth allows the add on to blend in as an integral part of the language and only HS/FORTH provides true multiple inheritance & membership.

Lets define classes BODY, ARM, and ROBOT, with methods MOVE and RAISE. The ROBOT class inherits:

```
INHERIT> BODY
HAS> ARM RightArm
HAS> ARM LeftArm
```

If Simon, Alvin, and Theodore are robots we could control them with:

```
Alvin 's RightArm RAISE          or:
+5 -10 Simon MOVE                or:
+5 +20 FOR-ALL ROBOT MOVE
```

Now that is a null learning curve!

**WAKE UP ! ! !**

Forth is no longer a language that tempts programmers with "great expectations", then frustrates them with the need to reinvent simple tools expected in any commercial language.

**HS/FORTH Meets Your Needs!**

Don't judge Forth by public domain products or ones from vendors primarily interested in consulting - they profit from not providing needed tools! Public domain versions are cheap - if your time is worthless. Useful in learning Forth's basics, they fail to show its true potential. Not to mention being s-l-o-w.

We don't shortchange you with promises. We provide implemented functions to help you complete your application quickly. And we ask you not to shortchange us by trying to save a few bucks using inadequate public domain or pirate versions. We worked hard coming up with the ideas that you now see sprouting up in other Forths. We won't throw in the towel, but the drain on resources delays the introduction of even better tools. Don't kid yourself, you are not just another drop in the bucket, your personal decision really does matter. In return, we'll provide you with the best tools money can buy.

**The only limit with Forth is your own imagination!**

You can't add extensibility to fossilized compilers. You are at the mercy of that language's vendor. You can easily add features from other languages to **HS/FORTH**. And using our automatic optimizer or learning a very little bit of assembly language makes your addition zip along as well as in the parent language.

Speaking of assembly language, learning it in a supportive Forth environment turns the learning curve into a light speed escalator. People who failed previous attempts to use assembly language, conquer it in a few hours or days using HS/FORTH.

**HS/FORTH** runs under MSDOS or PCDOS, or from ROM. Each level includes **all** features of lower ones. Level upgrades: $25. plus price difference between levels. Source code is in ordinary ASCII text files.

All HS/FORTH systems support full megabyte or larger programs & data, and run faster than any 64k limited ones even without automatic optimization -- which accepts almost anything and accelerates to near assembly language speed. Optimizer, assembler, and tools can load transiently. Resize segments, redefine words, eliminate headers without recompiling. Compile 79 and 83 Standard plus F83 programs.

**PERSONAL LEVEL $299.**
**NEW! Fast direct to video memory** text & scaled/clipped/windowed graphics in bit blit windows, mono, cga, ega, vga, all ellipsoids, splines, bezier curves, arcs, turtles; lightning fast pattern drawing even with irregular boundaries; powerful parsing, formatting, file and device I/O; DOS shells; interrupt handlers; call high level Forth from interrupts; single step trace, decompiler; music; compile 40,000 lines per minute, stacks; file search paths; format to strings. software floating point, trig, transcendental, 18 digit integer & scaled integer math; vars: A B * IS C compiles to 4 words, 1..4 dimension var arrays; **automatic optimizer for machine code speed.**

**PROFESSIONAL LEVEL $399.**
hardware floating point - data structures for all data types from simple thru complex 4D var arrays - operations complete thru complex hyperbolics; turnkey, seal; interactive dynamic linker for foreign subroutine libraries; round robin & interrupt driven multitaskers; dynamic string manager; file blocks, sector mapped blocks; x86&7 assemblers.

**PRODUCTION LEVEL $499.**
Metacompiler: DOS/ROM/direct/indirect; threaded systems start at 200 bytes, Forth cores from 2 kbytes; C data structures & struct+ compiler; TurboWindow-C MetaGraphics library, 200 graphic/window functions, PostScript style line attributes & fonts, viewports.

**ONLINE GLOSSARY $ 45.**

**PROFESSIONAL and PRODUCTION LEVEL EXTENSIONS:**

**FOOPS+** with multiple inheritance **$ 79.**
**TOOLS & TOYS DISK $ 79.**
**286FORTH** or **386FORTH $299.**
16 Megabyte physical address space or gigabyte virtual for programs and data; DOS & BIOS fully and freely available; 32 bit address/operand range with 386.
**ROMULUS** HS/FORTH from ROM **$ 99.**
**FFORTRAN** translator/mathpak **$ 79.**
Compile Fortran subroutines! Formulas, logic, do loops, arrays; matrix math, FFT, linear equations, random numbers.

Shipping/system: US: $7. Canada: $19. foreign: $49. We accept MC, VISA, & AmEx

*A message from the President of the Forth Interest Group*

# President's Letter

# I Have a Dream

**Forth Dimensions**

Although *Forth Dimensions* is the best and most beautiful publication that any language group in the world has ever produced, there is always room for more improvements. One area where new ideas in *Forth Dimensions* will complement the new directions we are taking FIG, is in the area of education: education of each of us, and education of members new to FIG and the Forth community.

*For each of us:* It has been suggested by Russell Harris that we need technical articles dealing with hands-on construction. Many of us would welcome more how-to-do-it articles. We need a "Steve Ciarcia" of *Forth Dimensions,* or at least an editor of a column along those lines. Russell was astounded that someone had passed up the opportunity to explain the construction of the "Forth Gizmo" from the 1988 Los Angeles Forth Convention in *Forth Dimensions.* He could be convinced to write some hands-on articles and serve as the editor of such a column, if others will also write "get-your-hands-dirty" articles. Marlin will be coordinating this, so start organizing your own contribution.

*For new members:* We need articles oriented toward new Forth users. We have become so sophisticated with Forth that we can only talk to each other. How many Forths are written these days in other languages, to demonstrate the equivalences to people who use those other languages? We write Forth these days in Forth. We wind up talking to ourselves. Each of us, no matter how new to Forth or how sophisticated, can write an article explaining some new insight that we just gained using Forth.

***From the days of the programming Michelangelos, we have reached the depths of the paint mixers.***

**Technical Hot Lines**

One thing that happened when the Forth Interest Group, Inc. separated from the Silicon Valley FIG Chapter was that we lost much of our technical Forth context and contacts. FIG supplies the indirect technical information, but we have lost our ability for direct face-to-face technical support. During a recent SVFIG meeting when this topic was discussed, many people seemed only to be waiting to be asked to provide their time and telephone for a Technical Hot Line. We have started a list of experts for the Silicon Valley area. We need to extend this idea to the rest of the world. If any of you would like to participate and be available to answer technical questions about Forth or Forth-related topics, we will establish in *FD* a list of experts on different topics with telephone numbers.

**I Have a Dream...**

We aren't a little group with little ideas. We are an organization of 1600 very devoted and very idealistic individuals. We have made an impact over the 13 years we have existed. What other computer language group that is devoted to such radical changes can say that? The dream is not dead. It is probably more alive than it has ever been. The general downturn in the economy has discouraged us. The pressure from "iron foundries" to sell larger and more expensive doodads has blinded us. The venture capitalists waving money in our faces have confused us.

The dream is still alive. All we have to do is to think bigger, to bypass and step over these obstacles. While we debated our navel (case statements, control structures, ANSI...), the rest of the software industry had time to catch up. We don't have the advantage we once had. We have become smug.

But one spark, and the whole world will flame Forth again. We have to get working in the new areas that have developed while we debated, in areas where we will again capture the imagination of the world. How many of us have worked with the new RISC processors? How many of us have used object-oriented programming methods (in its pure form, it is very close to our philosophy)? Have we seen a parallel Forth?

We can't afford to let opportunities pass us by, as we have in the past. How many of us supported F83 when it first became available? Did FIG endorse or promote it? How many of us immediately supported the Novix chip when it first appeared? So it had problems—with enough support, all problems can be fixed. How many of us have put our jobs on the line to put Forth into the systems we were building?

We think too small. The day-to-day blinders limit our scope of the world and restrict our dreams. Break out and dream again! We are only limited by our narrow view of what is possible. Just as hardware has a limit to its speed and size and is rapidly coming to that limit, software has reached the limit of its complexity. Even simple applications are

(Letters, from page 12.)

reaching the limit where complex languages and complex thinking are geometrically driving the necessary people and money beyond the reach of individuals.

I am tired of the complicated and complex, the dull and drab technologies, and work-for-wages technocrats. Programming is an art and will always be. From the days of the programming Michelangelos, we have reached the depths of the paint mixers.

I want to polish the simplicity and elegance and let the Sword of Forth cut through the Gordian Knots of COBOL, Fortran, and C.

*I Still Have a Small Dream...*

I want to make Forth one of the major programming philosophies of the world.

*I Have a Bigger Dream...*

I want to make the world better with applications that only Forth can make possible.

* * *

I am always available for comments (and maybe some humility).

*—John Hall*
*510-535-1294*
*JDHALL on GEnie*

about the previous or next standard or version. Because of economic and time constraints, too many defections from Forth will make anything about Forth irrelevant. If Forth has not been a widely accepted language anyway, why not have a fresh beginning, just the way FIG got started. With all the best programming minds in the industry, we should be able to attract new Forthians (especially software and hardware houses) if we become (finally?) unified.

Note: Imagine what would happen if the Windows 3.0 operating system is a Forth system with DOS as only one of its default tasks. Users communicate with it using plain language, graphics, or (for power users) object-oriented Forth. This is only possible provided the company doing it makes money and puts some of it toward creating the next money-making Forth product with mass-market appeal.

There are many more questions and statements than we have time for. If you would like them all, please let us know.

We do not understand why Forthians keep saying how great Forth is, how many great programmers we have, and that Forth can hold its own against any language—yet we are a dying breed. Could the answer be, "Money, money, money makes the world (and Forth) go round"?

Take my advice. Use *FD* to make money for all Forthians or soon there will be fewer than 2000 hardcore hobbyists (professionals?).

We are prepared to sell our hardware (documentation included) at the lowest price of any similar hardware you can find on the world-wide market, provided it is used in a Forth language project and, if possible, that the project is described in *FD* and a ten percent commission is paid to *FD* if it publishes the vendor's name. (Using the profit motive to get support.)

John N.S. Tse
Managing Director
Chrisma Technology, Ltd.
45, Genting Lane #07-01
Genting Warehouse Complex
Singapore 1334

*We thank Mr. Tse sincerely for his advice, because it is possible that a sweeping change, of the scope he suggests, may be exactly what FIG and* FD *need. His arguments are given greater strength by the coincidental-but-congruous contents of this issue's "Guest Editorial" and "President's Letter." I will forward these recommendations to FIG's Business Group and Board of Directors, and will include specific reference to them in a letter soliciting feedback from Forth vendors. Our readers' and vendors' responses to this material will surely influence how the FIG leadership regards it.*

*I can offer some preliminary, personal responses to a few of Mr. Tse's points.*

*1. I agree that a major challenge is to address a wider readership within our limited space. Practical, task-oriented articles are always sought after and, with the help mentioned in this issue's "President's Letter," we hope to have found a way to start getting them. To date, we also have too little tutorial material for new Forth users; and, when we have addressed primarily the middle-of-the-road Forth user, we have risked boring the corps of seasoned Forth experts who have kept us technically strong for all these years.*

*2. Plans are under way right now to publish Forth news and product announcements; the success of this department will rely on readers informing us of relevant items, and on vendors and developers sending timely press releases and announcements* (very *few currently do so).*

*4.* FD *currently will publish honest articles about Forth-drivable hardware of any kind, even if written by the developer, provided that it is not just an advertisement in disguise—that is, the technical information must be at least as valuable to our readers as the space it occupies on our pages. To go further and begin publishing "extended press releases" as short articles, written by parties with a vested interest in those products, would also be possible if* FD *receives a clear mandate to convert to an industry trade magazine format (see next item).*

*5. Certainly many of the changes suggested would require an entirely different philosophy on FIG's part. Changing a magazine's direction and format can be accomplished with simple (but not* that *simple) logistics; changing its supporters' beliefs, expectations, and desires for it is another thing altogether. Perhaps Mr. Tse is right—suddenly having a Forth industry trade publication might be heady stuff, properly invigorating, and good outreach to those we have not been able to address in our current format. But we would have to step out of the ivory tower of*

(Continued on page 31.)

# Best of GEnie

If you thought discussion regarding the pending Forth Standard was waning, think again! Several issues still remain unresolved and bear some serious thought before the book is closed and the seal is waxed on dp ANS Forth. One of the more unresolved of these issues is that of address alignment. Please read the exchanges captured June 20, 1990 from ForthNet ports off RIME and Usenet comp.lang.forth, and from GEnie Forth RoundTable participants in Category 10, Topic 25.

I have begun this discussion with a proposal presented by Jack Woehr regarding problems unique to implementors of embedded Forth systems. This proposal alone amplifies the X3J14 Technical Committee's task. It is not enough to make a set of rules—they must also consider how those rules affect a variety of platforms, not the least of which is embedded systems.

Read, discover, participate.

**Category 10: Forth Standards**

***Discussions about the ANS Forth Standard for Tick, >BODY, ['] ...***

*Magnet:Charles Keane*

X3J14 Proposal

*Title:* ROM-based Systems Quibble with >BODY

*Words:*
>BODY
CREATE DOES>
ENVIRONMENT?

*Abstract:*
>BODY as defined in BASIS 14 may benefit from redefinition with an eye to portability between mixed RAM/ROM and RAM-only systems.

*Proposal:*
8.1.0550 >BODY

In conjunction with 5.3.2 "Addressable Memory," this construct and the underlying concepts of PFA appear to be ambiguous for ROMmed creatures of CREATE which contain address tokens in their PFA.

Propose: "a-addr is the parameter field associated with the execution token w of a word defined via CREATE. The contents of this address may be constant data, such as an address token to memory where the data which makes the CREATEd word useful is stored (as is often the case in a ROMmed system), or such data itself (as is typically the case in a RAM-only system). If there is any question as to which is the case, a Standard program should compare the token returned by >BODY with the token returned by EXECUTEing the CREATEd word itself."

The counter argument could be brought that the above technique would not work for CREATE...DOES>. In such case, another CREATE construct could be examined by a Standard program to determine what sort of PFAs CREATE creates. In any event, it is hard to imagine a truly portable Standard program that would want access to the internals of a CREATE ...DOES> word via >BODY. It would be safer, in such cases, simply to create some data structure that was more easily manipulable and then to write a colon definition that performed the desired action upon it.

Alternatively, perhaps a query string could be defined for the ENVIRONMENT? construct (8.1.1345) which could inform the Standard program as to whether CREATE words contain data or address pointers in their PFAs.

Submitted by:
Jack J. Woehr
Vesta Technology Inc.
7100 W. 44th Ave, Suite #101
Wheat Ridge, Colorado 80033
Voice: (303) 422-8088
FAX: (303) 422-9800
BBS: (303) 278-0364
jax@well.UUCP JAX on GEnie

* * *

Subject: When to ALIGN

In general, you don't need to ALIGN before @ and !, but instead when using , (comma) after C,. It's usually used when creating data structures.

—Mitch Bradley
wmb@Eng.Sun.COM

* * *

Subject: ALIGN
Reply-To:
UNBCIC%
BRFAPESP.BITNET@
SCFVM.GSFC.NASA.GOV

> "...when you <BUILDS things, you need to align it. And, if the word DOES> nothing, the user will have to use ALIGN before @ and ! too. Actually, that's not true, if the system implementor did things right. The last word-aligned system I used automatically ALIGNed before every CREATE. This forced the parameter field to an even address (which was required for thread of a colon definition). So DOES> always returned an aligned address, and the user didn't have to worry about it.
>
> "Strings compiled inline were always padded to an even number of bytes; this required a small bit of additional logic in run-time code which advances the IP over the string, but it was invisible to the user. (Inline byte parameters were forbidden, no great loss.)"

1) I think the loss of the ability to compile bytes is a great loss.

2) How about:

```
: DATA CREATE
  ALLOT ( NAME ) ,
  ( AGE ) ;
  15 30 DATA NAME_1
```

Just putting 15 won't work. SPARCs have a four-bytes alignment restriction, too, for example. And on and on. And RECORD structures *are very useful.*

—Daniel C. Sobral
UNBCIC@BRFAPESP.BITNET

* * *

> "1) I think the loss of the ability to compile bytes is a great loss."

Well, in the system I was speaking of, you didn't lose that ability. Structures had no alignment restrictions other than *starting* at a word boundary. Which means that, yes, if you were careless, you could create a structure which would lead to an addressing violation.

What was lost was the ability to, for example, compile a BRANCH with a one-byte offset ("in-line" parameter). This was because the thread needed to maintain word alignment. You didn't lose any capabilities with this restriction, just some micro-optimizations of memory usage.

—Brad Rodriguez
B.RODRIGUEZ2 [Brad]

* * *

Subject: Addressability of data space
Reply-To:
Mitch.Bradley%
ENG.SUN.COM@
SCFVM.GSFC.NASA.GOV

> "The troublesome clause from BASIS13 is from section 5.3.2. It clearly states:
>
> "'*...it is an exception if a Standard Program addresses memory other than [in dictionary space regions] from the address provided by a CREATEd word or HERE to the end of the region generated by consecutive allocations (, , C, , ALLOT, ALIGN) made without intervening definitions or deallocations* (FORGET)...' [the rest of this section is about non-dictionary space]
>
> "This means that if you build a defined word with CREATE (or a word like DEFER which uses CREATE), say CREATE FOO, you can use the address returned by FOO. Period. Nowhere does it say you can tick FOO for its parameter field address, and this clause is carefully worded such that anything not explicitly permitted is forbidden.
>
> "Has this clause been fixed in the latest BASIS?"

Basis 15 says pretty much the same thing (it's now section 5.4).

I believe that this text is logically correct. The text says that memory at that address is addressable. It does not, and indeed cannot, enumerate all the possible ways of putting that address on the stack. For example, one could do the following:

```
CREATE FOO
1 C, 2 C, 3 C, 4 C,
5 C, HERE
CONSTANT XYZZY
7 XYZZY 5 - C!
```

The point is, section 5.4 says that the memory address provided by a CREATEd word and by HERE is addressable, and that other memory addresses are not addressable. It does *not* say that executing the CREATEd word is the only way of calculating that same address.

However, since this section has already been misunderstood, I would like to hear suggestions for how to improve the wording. I find that writing extremely precise English text is a very challenging task.

By the way, here's what Basis 15 says about >BODY:

*8.1.0550 >BODY*
*"to-body" CORE*
*( w -- a-addr )*
*a-addr is the data field address corresponding to the execution token w of a word defined via CREATE.*
*See also: 5.4 Addressable Memory*

The rationale box says:
*a-addr is the address that HERE would have returned had it been executed immediately after the execution of the CREATE that defined w.*

—Mitch Bradley
Mitch.Bradley@Eng.Sun.COM

* * *

Subject: Addressability of data space
Reply-To:
wbrown@beva.bev.lbl.gov
(Bill Brown)

Seems I recall hearing somewhere that somebody offers, or at least once upon a time offered, an 8052 with a version of Forth in on-board ROM. Does anybody know if it's still available, and if it is who sells it and for how much? I was sure that I had the details somewhere, however, if I do I must have put it in a *really safe place!*

My interest is triggered by an article in the May '91 issue of *Elektor Electronics USA* which has to do with an 8032/8052 single-board computer project. It mentions using an 8052 with BASIC in ROM, and at first glance it looks like it would make a neat Forth gadget, assuming that the Forth version of the 8052 is available.

Disclaimer: These opinions are my own and have nothing to do with the official policy or the management of Lawrence Berkeley Labs, who probably couldn't care less about employees who play with trains.

—Bill Brown
wbrown@beva.bev.lbl.gov

* * *

Okay, I have a copy of BASIS15 now. According to BASIS13, your example:

```
CREATE ...
HERE CONSTANT XYZZY
```

would not necessarily work, because nothing equated the address returned by CREATEd words to the address returned by HERE.

The first key addition in BASIS 15 was section 5.4.1, which states (among other things), "HERE always identifies the beginning of the next region to be allocated."

The second key addition was the rationale note in >BODY that you quoted (although I don't know if the rationale note carries the same weight as the text of the standard itself).

At any rate, you've answered my question—the problem was fixed in BASIS 15.

By the way, I found the

section in BASIS 13 perfectly understandable, Mitch. It's just that there was a difference between what it said and what everyone assumed. Thanks (to you or whoever) for elucidating this in BASIS 15.

—Brad Rodriguez
brad%candice@maccs.uucp
(God willing) *or*:
B.RODRIGUEZ2 on GEnie
*or*:
brad%candice@
maccs.dcss.mcmaster.ca
*or*:
bradford@
maccs.dcss.mcmaster.ca
(archaic)

* * *

Subject: I.2.4 Alignment Problems
Keywords: BASIS 15 ALIGN ALIGNED
Re: BASIS 15 I.2.4 Alignment Problems

> "An implementor of ANS Forth can handle these alignment restrictions in one of two ways. Forth memory access words (@, !, +!, etc.) could be implemented in terms of smaller width access instructions which have no alignment restrictions....
>
> "Although this conceals hardware ugliness from the programmer, it is inefficient.
>
> "An alternative implementation of ANS Forth could define each memory access word using the native instructions that most closely match the word's function....
>
> "In this case responsibility for giving @ a correctly aligned address devolves on the programmer.
>
> "A portable ANS Forth program must assume the worst case and use the alignment operators described below..."

The fundamental issue raised in Forth implementations on machine architectures with alignment restrictions, is whether to aim for maximum space efficiency (solution 1) or to aim for maximum speed efficiency (solution 2). Dependent on the kind of application, either of the solutions may result in better performance of a particular application. This suggests that the programmer (or even the user!) of the final application is best suited to make the space vs. speed decision. However, BASIS 15 leaves the decision to the implementor of the Forth system.

Big deal?!

Well, yes... because, in order to let the implementor make that decision, BASIS 15 supplies him with two core words (ALIGN and ALIGNED) that must be used by portable ANS Forth programs. Besides breaking existing code (already mentioned by Mitch Bradley), this "solution" places the alignment burden on *all programmers*, including those who do not use alignment-restricted hardware. Unfair would be the least to call this; in order to let some people have the advantage of a speedier Forth, all the rest should suffer from alignment indigestion.

But should we then force implementors to choose the first solution? In principle, yes, but this sounds worse than it actually is:

My suggestion would be for Forths on aligned machines to implement both the space- and the speed-efficient versions of the memory-access words. Furthermore, when dealing with the speed-efficient words, the character unit should be cell-size so every operator would keep addresses aligned.

Different word lists should be used for the two kinds of definitions; the space-efficient words could, for example, be kept in SMALL, whereas their speed-efficient counterparts would reside in FAST. Now, when a program is ported from a non-aligned to an aligned environment, the programmer can first select the appropriate versions by executing SMALL or FAST, resulting in either small or fast compiled code.

I'm sorry for LZ because he had to enter the whole alignedness into the basis document, but I would be even more sorry if hardware patches like ALIGN(ED) would enter the standard. For after all, who knows, in $x$ years alignment restrictions may no longer be relevant, but because some people in the 90s thought they were, Forthers are still aligning their data structures.

—Jan Stout
wsbusup4@rwa.urc.tue.nl
Eindhoven University of Technology, Netherlands

* * *

Subject: Address alignment
Reply-To: Mitch Bradley <Mitch.Bradley% ENG.SUN.COM@ SCFVM.GSFC.NASA.GOV>

In a threaded-code implementation, the penalty for arbitrary-alignment @ and ! operators is relatively small.

In an optimized native-code system, where you are really pushing for speed, the situation is somewhat different. @ and ! are often expanded in-line on those systems, and peephole optimization can frequently combine the access with nearby calculation steps and/or arithmetic and logical operators. The requirement for arbitrary alignment support makes this much more difficult, and the compiler is considerably less likely to succeed in generating excellent code.

I ran into this problem when I wrote a translator program that would convert 68000 binary code into SPARC binary code. 68000s are two-byte aligned, and SPARCs are four-byte aligned. The alignment problems made the generated SPARC code much worse in the general case, and caused me to go to a lot of trouble to get the translator to guess about actual alignment at compile time.

My experience with Forth programmers is that many of them want to be able to get the most out of their hardware, and are willing to go to a bit of extra programming effort to get it (e.g., by adding ALIGNED at judicious [places].

—Mitch Bradley
Mitch.Bradley@ENG.SUN.COM

* * *

Subject: Alignment
Reply-To: Mitch.Bradley% ENG.SUN.COM@ SCFVM.GSFC.NASA.GOV

> "Hmmm... reading Robert Berkey's comments, I'm beginning to believe that *all* existing Forth code will be rendered nonconforming by the BASIS."

In a sense, this is correct. However, I think a better way to look at it is as follows:

ANS Forth will not magically make existing code portable. Existing code will most likely continue to run on the same systems that it currently runs on. Existing code that assumes arbitrary alignment is currently not portable to implementations that do not choose to "hide" hardware alignment restrictions (a significant percentage of Forth implementations for such hardware).

—Mitch Bradley
Mitch.Bradley@ENG.SUN.COM
* * *

Subject: Align
Reply-To:
UNBCIC%
BRFAPESP.BITNET@
SCFVM.GSFC.NASA.GOV

"From: Rob Sciuk
"Subject: RE: Memory Management/PIC

"Elizabeth points out that any standard defining word should take care to align words (bodies, headers, and fields contained therein) on appropriate boundaries. Further, ALLOT and , should align on cell boundaries, and C, should ensure that the next invocation of HERE, ALLOT, , (comma), etc. will utilize a cell boundary appropriate to the processor [mine]."

C, should ensure that the next invocation of HERE, ALLOT, ... will utilize a *cell* boundary?! It's better to live with a slow @ and ! than with this! We have only two options:

1) Throw an overhead upon HERE, ALLOT, ...;
2) Make C, ALLOT a *cell, thus acting* as a comma.

Another thing, if ALLOT and HERE always return an aligned address, it's better to make this very clear in the standard, or Structure Wordsets (which are very common) will be a source of lots of errors. I wouldn't like an ALLOT that aligns, but then, you can never satisfy everyone.

Errare Humanum Est...
...Perseverare Autem Diabolicum
—Daniel C. Sobral
UNBCIC@BRFAPESP.BITNET
UNBCIC@FPSP.FAPESP.ANSP.BR
(No one but me is responsible for the above message.)
* * *

Subject: Align

Daniel C. Sobral writes:

"C, should ensure that the next invocation of HERE, ALLOT... will utilize a cell boundary?!"

Good, it wasn't just me who thought this was a lousy idea. I was wondering how C, would ever accomplish this, short of always allocating enough bytes to end up on a cell boundary. But then, how do you pack bytes with successions of C,?

I'm always hesitant of posting to this group; having read publications by many of the other posters, it is hard for me to think of myself as a peer. For example, I assume there must be something I don't understand about all these ALIGNment issues. Haven't we been living with ALIGN on 68000s for a decade now? I've always assumed that the implementation was pretty straightforward: ALLOT ensures that the address generated for the variable being allotted is appropriate to the size of the variable, allocating extra bytes to make it so. Of course, this assumes the size is a "natural" size for the processor, usually bytes, longs, etc. For "unnatural" records, you had to align things manually. Is there something new I'm missing?

By the way: alignment to a cell boundary is not necessarily sufficient, depending on the processor. For example, the i860 requires address alignment to be MOD (size of variable), or there is a very high performance penalty on memory accesses.

—Nicolas Tamburri
nick@sw.stratus.com
* * *

*—Gary Smith*
*GARY-S on GEnie*

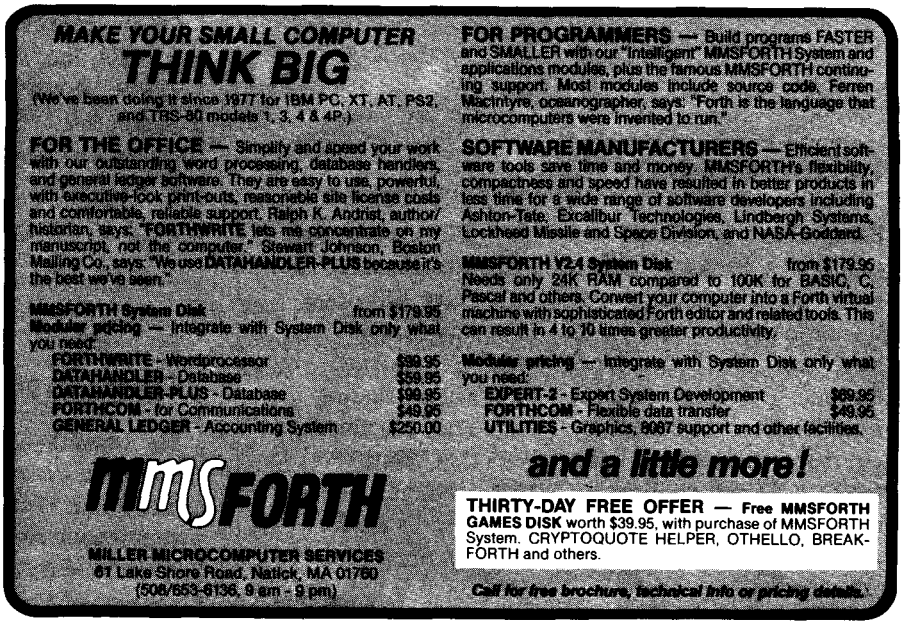
*(Letters, from page 27.)*

*technical objectivity and commercial impartiality (to the degree that we have achieved either of those), and relinquish the "clubby" familiar-ness of what we have enjoyed all these years. (See Mr. Tse's points ten, 11, 13, 14, 17).*

*7. Yes, please! We do want to publish examples of Forth doing a good job at an interesting task, or even at a boring task if it demonstrates technology that can be transferred profitably to other sites/applications by other Forth users.*

*These few thoughts, as I stated above, are preliminary—Mr. Tse's letter crossed my desk during final pre-press preparations. His remarks challenge us to open up to greater possibilities and higher stakes than we have considered here before. May they lead us to thoughtful consideration and fruitful discussion, and to a vision of Forth and FIG that will serve well in the future.*

*Please send your replies to me at the Forth Interest Group mailing address or to my MARLIN.O e-mail address on GEnie. —Ed.*

*Forth resources & contact information*

# reSource Listings

*Please send updates, corrections, additional listings, and suggestions to the Editor.*

## Forth Interest Group

The Forth Interest Group serves both expert and novice members with its network of chapters, *Forth Dimensions*, mail-order services, and on-line activities. For membership information, or to reserve advertising space, contact the administrative offices:

Forth Interest Group
P.O. Box 8231
San Jose, California 95155
408-277-0668
Fax: 408-286-8988

*Board of Directors*
John Hall, President
C.H. Ting, Vice-President
Mike Elola, Secretary
Dennis Ruffer, Treasurer
Wil Baden
Jack Brown
David Petty
Dennis Ruffer

*Founding Directors*
William Ragsdale
Kim Harris
Dave Boulton
Dave Kilbridge

## In Recognition

Recognition is offered annually to a person who has made an outstanding contribution in support of Forth and the Forth Interest Group. The individual is nominated and selected by previous recipients of the "FIGGY." Each receives an engraved award, and is named on a plaque in the administrative offices.

1979 William Ragsdale
1980 Kim Harris
1981 Dave Kilbridge
1982 Roy Martens
1983 John D. Hall
1984 Robert Reiling
1985 Thea Martin
1986 C.H. Ting
1987 Marlin Ouverson
1988 Dennis Ruffer
1989 Jan Shepherd
1990 Gary Smith

## ANS Forth

The following members of the ANS X3J14 Forth Standard Committee are available to personally carry your proposals and concerns to the committee. Please feel free to call or write to them directly:

Gary Betts
Unisyn
301 Main, penthouse #2
Longmont, CO 80501
303-924-9193

Charles Keane
Performance Pkgs., Inc.
515 Fourth Avenue
Watervleit, NY 12189-3703
518-274-4774

Mike Nemeth
CSC
10025 Locust St.
Glenndale, MD 20769
301-286-8313

George Shaw
Shaw Laboratories
P.O. Box 3471
Hayward, CA 94540-3471
415-276-5953

Andrew Kobziar
NCR
Medical Systems Group
950 Danby Rd.
Ithaca, NY 14850
607-273-5310

David C. Petty
Digitel
125 Cambridge Park Dr.
Cambridge, MA 02140-2311

Elizabeth D. Rather
FORTH, Inc.
111 N. Sepulveda Blvd.,
suite 300
Manhattan Beach, CA 90266
213-372-8493

## Forth Instruction

*Los Angeles*—Introductory and intermediate three-day intensive courses in Forth programming are offered monthly by Laboratory Microsystems. These hands-on courses are designed for engineers and programmers who need to become proficient in Forth in the least amount of time. Telephone 213-306-7412.

## On-Line Resources

*To communicate with these systems, set your modem and communication software to 300/1200/2400 baud with eight bits, no parity, and one stop bit, unless noted otherwise. GEnie requires local echo.*

*GEnie*
For information,
call 800-638-9636
- Forth RoundTable *(ForthNet*)*
  Call GEnie local node, then type M710 or FORTH
  SysOps:
  Dennis Ruffer (D.RUFFER), Scott Squires (S.W.SQUIRES), Leonard Morgenstern (NMORGENSTERN), Gary Smith (GARY-S)

- MACH2 RoundTable
  Type M450 or MACH2
  Palo Alto Shipping Company
  SysOp:
  Waymen Askey (D.MILEY)

*BIX (ByteNet)*
For information,
call 800-227-2983
- Forth Conference
  Access BIX via TymNet, then type j forth
  Type FORTH at the : prompt
  SysOp:
  Phil Wasson (PWASSON)

- LMI Conference
  Type LMI at the : prompt
  LMI products
  Host:
  Ray Duncan (RDUNCAN)

*CompuServe*
For information,
call 800-848-8990
- Creative Solutions Conf.
  Type !Go FORTH
  SysOps: Don Colburn, Zach Zachariah, Ward McFarland, Jon Bryan, Greg Guerin, John Baxter, John Jeppson

- Computer Language Magazine Conference
  Type !Go CLM
  SysOps: Jim Kyle, Jeff Brenton, Chip Rabinowitz, Regina Starr Ridley

*Unix BBS's with forth.conf (ForthNet* and reachable via StarLink node 9533 on TymNet and PC-Pursuit node casfa on TeleNet.)*
- WELL Forth conference
  Access WELL via CompuserveNet
  or 415-332-6106
  Fairwitness:
  Jack Woehr (jax)

- Wetware Forth conference
  415-753-5265
  Fairwitness:
  Gary Smith (gars)

*PC Board BBS's devoted to Forth (ForthNet*)*
- British Columbia Forth Board
  604-434-5886
  SysOp: Jack Brown

- Grapevine
  501-753-8121 to register
  501-753-6389
  StarLink node 9858
  SysOp: Jim Wenzel

- Real-Time Control Forth Board
  303-278-0364
  StarLink node 2584 on TymNet
  PC-Pursuit node coden on TeleNet
  SysOp: Jack Woehr

*Other Forth-specific BBS's*
- Laboratory Microsystems, Inc.
  213-306-3530
  StarLink node 9184 on TymNet
  PC-Pursuit node calan on TeleNet
  SysOp: Ray Duncan

- Knowledge-Based Systems
  Supports Fifth
  409-696-7055

- Druma Forth Board
  512-323-2402
  StarLink node 1306 on TymNet
  SysOps: S. Suresh, James Martin, Anne Moore

*Non-Forth-specific BBS's with extensive Forth libraries*
- DataBit
  Alexandria, VA
  703-719-9648
  PCPursuit node dcwas
  StarLink node 2262
  SysOp: Ken Flower

- The Cave
  San Jose, CA
  408-259-8098
  PCPursuit node casjo
  StarLink node 6450
  SysOp: Roger Lee

*International Forth BBS's*
- Melbourne FIG Chapter
  (03) 809-1787 in Australia
  61-3-809-1787 international
  SysOp: Lance Collins

- Forth BBS JEDI
  Paris, France
  33 36 43 15 15
  7 data bits, 1 stop, even parity

- Max BBS *(ForthNet*)*
  United Kingdom
  0905 754157
  SysOp: Jon Brooks

- Sky Port *(ForthNet*)*
  United Kingdom
  44-1-294-1006
  SysOp: Andy Brimson

- SweFIG
  Per Alm Sweden
  46-8-71-35751

- NEXUS Servicios de Informacion, S. L.
  Travesera de Dalt, 104-106, Entlo. 4–5
  08024 Barcelona, Spain
  + 34 3 2103355 (voice)
  + 34 3 2147262 (modem)
  SysOps: Jesus Consuegra, Juanma Barranquero
  barran@nexus.nsi.es (preferred)
  barran@nsi.es
  barran (on BIX)

**ForthNet is a virtual Forth network that links designated message bases in an attempt to provide greater information distribution to the Forth users served. It is provided courtesy of the SysOps of its various links.*

*This list was accurate as of February 1991. If you know another on-line Forth resource, please let me know so it can be included in this list. I can be reached in the following ways:*

*Gary Smith*
*P. O. Drawer 7680*
*Little Rock, Arkansas 72217*
*Telephone: 501-227-7817*
*Fax (group 3): 501-228-9374*
*GEnie (co-SysOp, Forth RT and Unix RT): GARY-S*
*Usenet domain.: uunet!ddi1!lrark!glsrk!gars*

## FIG Chapters

*The Forth Interest Group Chapters listed below are currently registered as active with regular meetings. If your chapter listing is missing or incorrect, please contact Anna Brereton at the FIG office's Chapter Desk. This listing will be updated regularly in* Forth Dimensions. *If you would like to begin a FIG Chapter in your area, write for a "Chapter Kit and Application."*

*Forth Interest Group*
*P.O. Box 8231*
*San Jose, California 95155*

### U.S.A.

- **ALABAMA**
  **Huntsville Chapter**
  Tom Konantz
  (205) 881-6483

- **ALASKA**
  **Kodiak Area Chapter**
  Ric Shepard
  Box 1344
  Kodiak, Alaska 99615

- **ARIZONA**
  **Phoenix Chapter**
  4th Thurs., 7:30 p.m.
  Arizona State Univ.
  Memorial Union, 2nd floor
  Dennis L. Wilson
  (602) 381-1146

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

  **North Bay Chapter**
  2nd Sat.
  12 noon tutorial, 1 p.m. Forth
  2055 Center St., Berkeley
  Leonard Morgenstern
  (415) 376-5241

  **Orange County Chapter**
  4th Wed., 7 p.m.
  Fullerton Savings
  Huntington Beach
  Noshir Jesung (714) 842-3032

  **Sacramento Chapter**
  4th Wed., 7 p.m.
  1708-59th St., Room A
  Bob Nash
  (916) 487-2044

  **San Diego Chapter**
  Thursdays, 12 Noon
  Guy Kelly (619) 454-1307

  **Silicon Valley Chapter**
  4th Sat., 10 a.m.
  Applied Bio Systems
  Foster City
  John Hall
  (415) 535-1294

  **Stockton Chapter**
  Doug Dillon (209) 931-2448

- **COLORADO**
  **Denver Chapter**
  1st Mon., 7 p.m.
  Clifford King (303) 693-3413

- **FLORIDA**
  **Orlando Chapter**
  Every other Wed., 8 p.m.
  Herman B. Gibson
  (305) 855-4790

- **GEORGIA**
  **Atlanta Chapter**
  3rd Tues., 7 p.m.
  Emprise Corp., Marietta
  Don Schrader (404) 428-0811

- **ILLINOIS**
  **Cache Forth Chapter**
  Oak Park
  Clyde W. Phillips, Jr.
  (708) 713-5365

  **Central Illinois Chapter**
  Champaign
  Robert Illyes (217) 359-6039

- **INDIANA**
  **Fort Wayne Chapter**
  2nd Tues., 7 p.m.
  I/P Univ. Campus
  B71 Neff Hall
  Blair MacDermid
  (219) 749-2042

- **IOWA**
  **Central Iowa FIG Chapter**
  1st Tues., 7:30 p.m.
  Iowa State Univ.
  214 Comp. Sci.
  Rodrick Eldridge
  (515) 294-5659

  **Fairfield FIG Chapter**
  4th Day, 8:15 p.m.
  Gurdy Leete (515) 472-7782

- **MARYLAND**
  MDFIG
  3rd Wed., 6:30 p.m.
  JHU/APL, Bldg. 1
  Parsons Auditorium
  Mike Nemeth
  (301) 262-8140 (eves.)

- **MASSACHUSETTS**
  **Boston FIG**
  3rd Wed., 7 p.m.
  Bull HN
  300 Concord Rd., Billerica
  Gary Chanson (617) 527-7206

- **MICHIGAN**
  **Detroit/Ann Arbor Area**
  Bill Walters
  (313) 731-9660
  (313) 861-6465 (eves.)

- **MINNESOTA**
  **MNFIG Chapter**
  Minneapolis
  Fred Olson
  (612) 588-9532

- **MISSOURI**
  **Kansas City Chapter**
  4th Tues., 7 p.m.
  Midwest Research Institute
  MAG Conference Center
  Linus Orth (913) 236-9189

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

- **NEW JERSEY**
  **New Jersey Chapter**
  Rutgers Univ., Piscataway
  Nicholas G. Lordi
  (908) 932-2662

- **NEW MEXICO**
  **Albuquerque Chapter**
  1st Thurs., 7:30 p.m.
  Physics & Astronomy Bldg.
  Univ. of New Mexico
  Jon Bryan (505) 298-3292

- **NEW YORK**
  **Long Island Chapter**
  3rd Thurs., 7:30 p.m.
  Brookhaven National Lab
  AGS dept.,
  bldg. 911, lab rm. A-202
  Irving Montanez
  (516) 282-2540

  **Rochester Chapter**
  Monroe Comm. College
  Bldg. 7, Rm. 102
  Frank Lanzafame
  (716) 482-3398

- **OHIO**
  **Columbus FIG Chapter**
  4th Tues.
  Kal-Kan Foods, Inc.
  5115 Fisher Road
  Terry Webb
  (614) 878-7241

  **Dayton Chapter**
  2nd Tues. & 4th Wed., 6:30 p.m.
  CFC
  11 W. Monument Ave. #612
  Gary Ganger (513) 849-1483

- **PENNSYLVANIA**
  Villanova Univ. Chapter
  1st Mon., 7:30 p.m.
  Villanova University
  Dennis Clark
  (215) 860-0700

- **TENNESSEE**
  **East Tennessee Chapter**
  Oak Ridge
  3rd Wed., 7 p.m.
  Sci. Appl. Int'l. Corp., 8th Fl.
  800 Oak Ridge Turnpike
  Richard Secrist (615) 483-7242

- **TEXAS**
  **Austin Chapter**
  Matt Lawrence
  PO Box 180409
  Austin, TX 78718

  **Dallas Chapter**
  4th Thurs., 7:30 p.m.
  Texas Instruments
  13500 N. Central Expwy.
  Semiconductor Cafeteria
  Conference Room A
  Warren Bean (214) 480-3115

  **Houston Chapter**
  3rd Mon., 7:30 p.m.
  Houston Area League of
  PC Users (HAL-PC)
  1200 Post Oak Rd.
  (Galleria area)
  Russell Harris
  (713) 461-1618

- **VERMONT**
  **Vermont Chapter**
  Vergennes
  3rd Mon., 7:30 p.m.
  Vergennes Union High School
  RM 210, Monkton Rd.
  Hal Clark (802) 453-4442

- **VIRGINIA**
  **First Forth of Hampton Roads**
  William Edmonds
  (804) 898-4099

  **Potomac FIG**
  D.C. & Northern Virginia
  1st Tues.
  Lee Recreation Center
  5722 Lee Hwy., Arlington
  Joseph Brown
  (703) 471-4409
  E. Coast Forth Board
  (703) 442-8695

  **Richmond Forth Group**
  2nd Wed., 7 p.m.
  154 Business School
  Univ. of Richmond
  Donald A. Full
  (804) 739-3623

- **WISCONSIN**
  **Lake Superior Chapter**
  2nd Fri., 7:30 p.m.
  1219 N. 21st St., Superior
  Allen Anway (715) 394-4061

## INTERNATIONAL

- **AUSTRALIA**
  **Melbourne Chapter**
  1st Fri., 8 p.m.
  Lance Collins
  65 Martin Road
  Glen Iris, Victoria 3146
  03/889-2600
  BBS: 61 3 809 1787

  **Sydney Chapter**
  2nd Fri., 7 p.m.
  John Goodsell Bldg., RM LG19
  Univ. of New South Wales
  Peter Tregeagle
  10 Binda Rd.
  Yowie Bay 2228
  02/524-7490
  Usenet:
  tedr@usage.csd.unsw.oz

- **BELGIUM**
  **Belgium Chapter**
  4th Wed., 8 p.m.
  Luk Van Loock
  Lariksdreef 20
  2120 Schoten
  03/658-6343

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

- **CANADA**
  **Forth-BC**
  1st Thurs., 7:30 p.m.
  BCIT, 3700 Willingdon Ave.
  BBY, Rm. 1A-324
  Jack W. Brown
  (604) 596-9764 or
  (604) 436-0443
  BCFB BBS (604) 434-5886

  **Northern Alberta Chapter**
  4th Thurs., 7–9:30 p.m.
  N. Alta. Inst. of Tech.
  Tony Van Muyden
  (403) 486-6666 (days)
  (403) 962-2203 (eves.)

  **Southern Ontario Chapter**
  Quarterly: 1st Sat. of Mar., June, and Dec. 2nd Sat. of Sept.
  Genl. Sci. Bldg., RM 212
  McMaster University
  Dr. N. Solntseff
  (416) 525-9140 x3443

- **ENGLAND**
  **Forth Interest Group-UK**
  London
  1st Thurs., 7 p.m.
  Polytechnic of South Bank
  RM 408
  Borough Rd.
  D.J. Neale
  58 Woodland Way
  Morden, Surry SM4 4DS

- **FINLAND**
  **FinFIG**
  Janne Kotiranta
  Arkkitehdinkatu 38 c 39
  33720 Tampere
  +358-31-184246

- **GERMANY**
  **Germany FIG Chapter**
  Heinz Schnitter
  Forth-Gesellschaft e.V.
  Postfach 1110
  D-8044 Unterschleissheim
  (49) (89) 317 3784
  e-mail uucp:
  secretary@forthev.UUCP
  Internet:
  secretary@Admin.FORTH-eV.de

- **HOLLAND**
  **Holland Chapter**
  Maurits Wijzenbeek
  Nieuwendammerdijk 254
  1025 LX Amsterdam
  The Netherlands
  ++(20) 636 2343

- **JAPAN**
  **Japan Chapter**
  Toshio Inoue
  University of Tokyo
  Dept. of Mineral Development
  Faculty of Engineering
  7-3-1 Hongo, Bunkyo-ku
  Tokyo 113, Japan
  (81)3-3812-2111 ext. 7073

- **REPUBLIC OF CHINA**
  **R.O.C. Chapter**
  Ching-Tang Tseng
  P.O. Box 28
  Longtan, Taoyuan, Taiwan
  (03) 4798925

- **SWEDEN**
  **SweFIG**
  Per Alm
  46/8-929631

- **SWITZERLAND**
  **Swiss Chapter**
  Max Hugelshofer
  Industrieberatung
  Ziberstrasse 6
  8152 Opfikon
  01 810 9289

***"We have to get working in the new areas that have developed while we debated, in areas where we will again capture the imagination of the world."***

*See "President's Letter"*

- **ITALY**
  **FIG Italia**
  Marco Tausel
  Via Gerolamo Forni 48
  20161 Milano

## SPECIAL GROUPS

- **Forth Engines Users Group**
  John Carpenter
  1698 Villa St.
  Mountain View, CA 94041

# Contributions from the Forth Community

We are beginning to assemble a great collection of Forth code in machine-readable form.
If you need a good Forth, it is probably here.

| | |
|---|---|
| Minimum-requirement Forths: | **PocketForth, PYGMY, eForth** |
| The kitchen-sink Forths: | **F-PC, BBL** |
| Complete starters: | **F83, Kforth, ForST** |
| Object-oriented Forths: | **Yerkes, MOPS** |
| Macintosh Forths: | **Yerkes, MOPS, PocketForth** |
| IBM Forths: | **PYGMY, F-PC, BBL, F83, Kforth, eForth** |
| Atari Forth: | **ForST** |
| 8051 Forths: | **8051 ROMmable Forth, eForth** |
| Graphic and floating-point Forths: | **Yerkes, MOPS, F-PC, Kforth** |
| Forth tutorials: | **The Forth Course, F-PC Teach** |
| Applications: | **Forth List Handler, Forth Spreadsheet, Automatic Structure Charts, A Simple Inference Engine, The Math Toolbox** |
| Great demos from St. Petersburg: | **AstroForth and AstroOKO** |

(See the Mail Order Form inside for more complete descriptions)

Yet to come:

- Collections of tools and techniques are being assembled that cover communications, hardware drivers, data analysis, and more math and numerical recipes.

Things we need or which are not currently available in machine-readable form:

- Original listings of fig-Forth for any machine on disk. We do not currently have them.
- We can use many more applications and application ideas that include source code.
- Code from the authors of FORML papers and past *Forth Dimensions* articles.

Send submissions to: **FIG, c/o Publications Committee, P.O Box 8231, San Jose, CA 95155**

**Forth Interest Group**
P.O.Box 8231
San Jose, CA 95155

Second Class
Postage Paid at
San Jose, CA