HIGH-LEVEL SINGLE-STEPPER

FORMATTING SOURCE CODE

TIME-KEEPING ROUTINE

CAPTURE
SC/FOX™ Forth Optimized eXpress™

SC/FOX PCS Parallel Coprocessor System
Uses Harris RTX 2000™ real-time Forth CPU.
System speeds options: 8 or 10 MHz.
Full-length 8 or 16-bit PC plug-in board.
64K to 1M bytes, 0 wait state static RAM.
Hardware expansion, 2 50-pin strip headers.
Operates concurrently with PC host.
Multiple PCS board parallel operation.
Memory accessible by PC host.
Communication thru PC I/O memory space.
Data transfer thru common 16K window.
Includes FCompiler, SC/Forth optional.
Prices start at $1,995 with software.

SC/FOX SBC Single Board Computer
Uses RTX 2000 real-time Forth CPU.
System speed options: 8, 10, or 12 MHz.
64K to 512K bytes 0-wait state static RAM.
64K bytes of shadow-EPROM space.
Application boot loader in EPROM.
Code converter for EPROM programs.
RS232 serial port with handshaking.
Centronic parallel-printer port.
Single +5 volt board operation.
Two 50-pin application headers.
Eurocard size: 100mm by 160mm.
Includes FCompiler, SC/Forth optional.
Prices start at $1,295 with software.

SC/Forth™ Language v3.0
Interactive Forth-83 Standard.
Hardware supported multitasking.
15-priority time-sliced multitasking.
Supports user-defined PAUSE.
Turnkey application support.
Extended control structures.
Double number extensions.
Infixed equation notation option.
Block or text file interpretation.
Word vectoring and module support.
Streamed instructions and recursion.
Automatic RTX 2000 optimizing compiler.
Microcode definitions and case statement.
Available for both SC/FOX PCS and SBC.
Price $995.

SC/FOX Support Products:
SC/Float™ IEEE Floating Point Library
SC/PCS/PROTO Prototype Board
SC/SBC/PROTO Prototype Board
SC/FOX/SP Serial-Parallel Board
XRUN™ Utilities
SC/SBC Serial Cable

Harris RTX 2000 Real-Time Forth CPU
1-cycle 16 x 16, 32-bit multiply.
1-cycle 14-prioritized interrupts.
One non-maskable interrupt.
Two 256-word stack memories.
Three 16-bit timer/counters.
8-channel multiplexed 16-bit I/O bus.
CMOS, 85-pin ceramic PGA package.

Ideal for embedded real-time control, high-speed data acquisition and reduction, image or signal processing, or computation intense applications. For additional information, please contact us at:

Silicon Composers, Inc., 210 California Avenue, Suite K, Palo Alto, CA 94306 (415) 322-8763
Forth source code was first written in a series of blocks, with occasional comments enclosed in parentheses. Later, further comments were sometimes included in shadow screens. But if source code requires explanation in order for us to use and maintain it, perhaps one should simply write all source code in the narrative form. Just indicate which parts of the narrative should be compiled; the rest is easily readable documentation.

Author Bacon gives us the definition of trouble and an answer to it, too. If a new Forth definition fails to produce the intended effects, the problem may be found with a single-stepper. This is not a program you can just type in and run. The strategy is to simulate the Forth interpreter and, since interpreters behave differently, you'll have to learn how your Forth interpreter works. Roll up your sleeves!

Sometimes the only way to learn Forth is to use it. With some encouragement from his son, the author chose to adapt a game originally published in BASIC. Although written by a neophyte, this version is much faster than its ancestor. In fact, the author put a timing loop into the code to give you a fair chance. Try to trap the deadly, stalking beasts...

Goto-less programs using "came from" sprang to our author's mind again when a program kept trying to execute variables whose base pointer had not been initialized. Most of the CFAs were zero so, like a bad joke, his system rebooted every time. He probably won't be the last to fool with it, but this EXECUTE will now check to see if a CFA is at least reasonable.

You can benefit from this program, which makes it routine to keep track of your time, even if you don't need to strictly audit your hours. It records the length of your current session at the computer, the total time for the day, and totals for the current and previous periods. The timelog file holds almost 50 pairs of login and logout times, so even down time and coffee breaks can be included.
Forth Interest Group
The Forth Interest Group serves both expert and novice members with its network of chapters, Forth Dimensions, and conferences that regularly attract participants from around the world. 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

Board of Directors
Robert Reiling, President (ret. director)
Dennis Ruffer, Vice-President
John D. Hall, Treasurer
Terri Sutton, Secretary
Wil Baden
Jack Brown
Mike Elola
Robert L. Smith

Founding Directors
William Ragsdale
Kim Harris
Dave Boulton
Dave Kilbridge
John James

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

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 link*)
  Call GEnie local node, then type M710 or FORTH
  SysOps: Dennis Ruffer (D.RUFFER), Scott Squires (S.W.SQUIRES), Leona 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 TymeNet, then type j forth
  Type FORTH at the : prompt
  SysOp: Phil Wasson (PWASSON)
- LMI Conference
  Type LMI at the : prompt
  Laboratory Microsystems products
  Host: Ray Duncan (RDUNCAN)

CompuServe
For information, call 800-848-8990
- Creative Solutions Conference
  Type !Go FORTH
  SysOps: Don Colburn, Zach Zachariah, 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 conferences (ForthNet links*)
- 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 links*)
- East Coast Forth Board
  703-442-8695
  SysOp: Jerry Schifrin
- North Coast Forth Board
  612-483-6711
  SysOp: Don Madson
- British Columbia Forth Board
  604-434-5886
  SysOp: Jack Brown
- Real-Time Control Forth Board
  303-278-0364
  SysOp: Jack Woehr

(Continued on page 37.)
EDITORIAL

I recently got this word from Nathaniel Grossman: "It appears that a fulminating debate on Forth programming style has been waiting to erupt." Yes, that ugly monster has reared its ill-documented head again. Have you noticed how everyone approves of code with good style, but resists being saddled with coding conventions? The former is something everyone is pretty sure they already have (and therefore approve of), while the latter rub a Forth programmer's independent streak the wrong way.

Our retrospective look at last year's real-time programming convention included Jef Raskin's remark that Forth's use of parentheses is backwards. He said Forth code could be inserted parenthetically as the working part of a human-language essay that clearly describes the task being performed. In a letter to the editor printed in the same issue, Bill Kibler wrote of the continuing discussions in his company about Forth style. He described some of the real costs of code that does not conform to a standard of style.

Even before that issue was printed, Glen Haydon offered his own timely contribution to the subject. Here you will find his code to let you embed Forth programs in clearly written (knock on wood) English descriptions of their function. Have a look at Glen's example, try it on some of your own code, then let us know your reactions, opinions, and variations. Also see Peter Verhoeff's time-keeping routine for a couple of incidental contributions to the discussion of style.

* * * * *

If coding style seems like weak fare, perhaps you should make reservations to attend euroFORML '89 on October 13-15 in West Germany. The theme will be loosely organized around real-time applications with, typically, plenty of leeway for digression to other Forth topics. The modern Hotel Selau is in a town near the village of Forth, about 20 km. west of Nuremberg. The conference languages will be English and Forth, and the proceedings will be published (deadline for included papers: October 1). Contact Marina Kern, Roter Hahn 42, D-2000 Hamburg 72, West Germany. Then come to the original FORML conference the next month in California to share what transpired!

* * * * *

Like a dental probe, discussion of a sensitive issue can make one wince; the closer to the problem, the stronger the reaction. Remember the HAL 9000? In the years since that fictitious AI entity's breakdown, psychologists have confirmed that things known but not acknowledged can, indeed, hinder problem solving. This issue's FIG chapters column may make you wince, but it may move you to become part of a strong local chapter near you and, thus, part of a stronger Forth Interest Group.

As the printed voice of that membership organization, Forth Dimensions intends to represent its readers, not to decide for them which topics are important or how Forth should be written. The articles and essays we publish are a slice of the Forth commonwealth. The collective domain expands and shifts, and so does the material we print.

Our publishing process is meant to foster mutual benefit and cooperative effort. But this means that, more than most magazines, we rely on our readers' input. Letters to the editor are a great way to join the dialog or to offer an improvement to someone's ideas. And having an article of your own printed in these pages brings attention and recognition from an audience of intelligent peers. We welcome the contribution you may make to our next volume, and hope to hear from you soon!

—Marlin Ouverson
Editor
Vintage Hardware
Dear Marlin,

I was pleased to read your comments about previous-generation machines (FD V/5). I have, since the first day of my introduction to computing, been working with either a “home computer” (TI 99/4A) or a PG. Mine is a Kaypro I with graphics screen and double-sided disks. I can’t justify the expense of a common PC, let alone point out that all the code was developed on even if the machine is an old one. I see, as occasional piece that is machine at hand.

It can’t do and do fast enough for the task have no call to ask anything of my 280 that something really current. That is because I published code in six articles and eight letters (albeit only one or two were machine specific).

I see no reason for you not to publish an occasional piece that is machine specific— even if the machine is an old one. I see, as you pointed out, good reason to publish for PGs as well as for current generation PCs and beyond.

Thanks for all the good work.

Sincerely,
Gene Thomas
4300 Bowman Road #103
Little Rock, AR 72210

Dear Marlin,

Kudos for including and eloquently justifying the Apple-specific article. I don’t and haven’t used one, but it was refreshing to see an alternative to the snootiness and disdain focused on us who stuck with our old stuff because it did what we wanted well enough to justify not rushing to buy the latest, fastest, whiz-bang Belchfire 500 as it came out.

On the phrase, “The rest is silence.” You asked Michael Perry if it came from The Hitchhiker’s Guide to the Galaxy and he denied it. Not surprising. The line pops up in HAIR, sung by the chorus in “The Flesh Failures (Let the Sunshine In”). One can also find:

“O, I die, Horatio;
The potent poison quite o’er-crows my spirit:
I cannot live to hear the news from England;
But I do prophesy the election ‘lights
On Fortinbras: he has my dying voice;
So tell him, with the occurrences, more and less,
Which have solicited.—The rest is silence.”

[Dies.]

—Hamlet, Act V, Scene II

Regards,
Glenn Toennes

More on Behalf of Wm. Shkspr.:

Dear Mr. Ouverson:

I am writing to provide you with the original attribution for the phrase, “The rest is silence,” which you mentioned in the editorial of the January/February issue. The phrase comes from near the end of Shakespeare’s tragedy Hamlet, Prince of Denmark, Act V, Scene II, line 370 (or thereabout, depending on the version).

I hope this helps.

Sincerely,
Robert Lee Hoffpauer

Mr. Hoffpauer also provided the same excerpt quoted above, and further jogged my memory with Horatio’s over-quoted response: Now cracks a noble heart. Good night, sweet prince; And flights of angels sing thee to thy rest!

I knew that.—Ed.

Forth Style

[The following is from a letter sent to Bill Kibler in response to his letter published in our last issue; its author kindly sent me a copy as of possible interest. In light of the current discussions about Forth style, I found it illuminating and encouraging.—Ed.]

Dear Mr. Kibler,

The latest Forth Dimensions arrived here yesterday, and I was able to read your letter about Forth programming style. It’s, of course, not a new concern; Kim Harris wrote a major article about it many years ago, and since the start of Forth publishing there have been articles appearing regularly.

I’ve just written a little article which is concerned, in part, with Forth programming style. It’s been submitted to Forth Dimensions. Of course, there is no “Forth style” as such, just as there is no unique or even dominant literary style in English. If all Forthwriters produced only code for embedded systems that had to fit into 4K, there might be a chance for a consensus style. If Forth were only for building Rapid-Files or VP-Planners, another style might appear. Then an enterprising Forth adept might produce a Forth analog of Grammatik II and impose a universal style.

Nevertheless, we all seem to agree that the principal need is to make Forth code readable by others—even by its own author. The actual mechanism for doing this may very well be implementation dependent. After all, we shouldn’t expect a screen-bound Forth implementation to host a commenting style that is stream-file friendly. A Forthwriter work-
HS/FORTH

YES, THERE IS A BETTER WAY
A FORTH THAT ACTUALLY
DELIVERS ON THE PROMISE

POWER

HS/FORTH's compilation and execution speeds are unsurpassed. Compiling at 20,000 lines per minute, it compiles faster than many systems link. For real jobs execution speed is unsurpassed as well. Even non-optimized programs run as fast as ones produced by most C compilers. Forth system design and tool benchmarks are slightly faster on nearly empty do loops, but bog down when the Colon nesting level approaches anything useful, and have much greater memory overhead for each definition. Our optimizer gives assembler language performance even for deeply nested definitions containing complex data and control structures.

HS/FORTH provides the best architecture, so good that another major vendor "cloned" (rather poorly) many of its features. Our Forth uses all available memory for both program and stack, with your choice of execution time penalty, and very little memory overhead. None at all for programs smaller than 200k. And you can re-size segments anytime, without a system reset. With the GigoForth option, your programs transparently enter native mode and expand into 16 Meg extended memory or a gigabyte of virtual, and run almost as fast as in real mode.

Benefits beyond speed and program size include word redefinition at any time and vocabulary structures that can be changed at will, for instance from simple to hashed, or from 79 Standard to Forth 83. You can re-define words names and reclaim space at any time. This includes automatic removal of a colon definition's local words become incredibly powerful, with points not requiring jumps over word fragments. One of many reasons our system is much more compact than its immense dictionary (16000 words) would imply.

INCREDIBLE FLEXIBILITY

The Rosetta Stone Dynamic Linker opens the world of utility libraries. Link to resident routines or link & remove routines interactively. HS/FORTH preserves relocatability of loaded libraries. Link to BTRIEVE METAWINDOWS HALO HOOPS as infinitum. Our call and data structure words provide easy linkage.

HS/FORTH runs both 79 Standard and Forth 83 programs, and has extensions covering vocabulary search order and the complete Forth 83 test suite. It loads and runs all FIG Libraries, the main difference being they load and run faster, and you can develop larger applications than with any other system. We like source code in text files, but support both file and sector mapped Forth block interfaces. Both line and block file loading can be nested to any depth and includes automatic path search.

FUNCTIONALITY

More important than how fast a system executes, is whether it can do the job at all. Can it work with your computer? Can it work with your other tools? Can it transform your data into answers. A language should be complete on the first two, and minimize the unavoidable effort required for the last.

HS/FORTH opens your computer like no other language. You can execute function calls, DOS commands, other programs interactively, from definitions, or even from files being loaded. DOS and BIOS function calls are well documented HS/FORTH words, we don't settle for giving you an INTCALL and saying "have at it". We also include both fatal and informative DOS error handlers, installed by executing FATAL or INFORM.

HS/FORTH supports character or blocked, sequential or random I/O. The character stream can be received from stdin to console, file, memory, printer or com port. We include a communications plus upload and download utility, and foreground background music. Display output through BIOS for compatibility or memory mapped for speed.

Our formatting and parsing words are without equal. Integer, double, quad, financial, scaled, time, date, floating or exponential, all our output words have string formatting counterparts for building records. We also provide words to parse all data types with your choice of field definition. HS/FORTH parses files from any language. Other words treat files like memory, read or write or from to a handle (file or device) as fast as possible. For advanced file support, HS/FORTH easily links to BTRIEVE, etc.

HS/FORTH supports text/graphic windows for MONO thru VGA. Graphic drawings (line rectangle ellipse) can be absolute or scaled to current window size and clipped, and work with our penplot routines. While great for plotting and line drawing, it doesn't approach the capabilities of Metawindows (tm Metagraphics). We use our Rosetta Stone Dynamic Linker to interface to Metawindows. HS/FORTH with MetaWindows makes an unbeatable graphics system. Or Rosetta to your own preferred graphics driver.

HS/FORTH provides hardware software floating point, including trig and transcendentals. Hardware fp covers full range int, log, exponential functions plus complex and hyperbolic counterparts, and all stack and combination ops. HS/FORTH supports all 8087 data types and works in RADIANs or DEGREES mode. No coprocessor needed. Operators (mostly fast machine code) and parse/word format controls cover numbers through 18 digits. Software fp eliminates conversion round off error and minimizes conversion time.

Single element through 4D arrays for all data types including complex use multiple cfa's to improve both performance and compactness. \[ Z = (X-Y) \] (X + Y) would be coded: X Y - Y + Z 16 (16 bytes) instead of: X @ Y @ - X @ Y @ + / Z 16 (26 bytes) Arrays can ignore 64k boundaries. Words use SYNCHRNYS for data type independence. HS/FORTH can even prompt the user for retry on erroneous numeric input.

The HS/FORTH machine coded string library with up to 3D arrays is without equal. Segment spanning dynamic string support includes insert, delete, add, find, replace, exchange, save and restore string storage.

We offer minimal overhead round robin and time slice multitaskers require a word that exits cleanly at the end of subtask execution. The cooperative round robin multitasker provides individual user stack segments as well as user tables. Control passes to the next task/user whenever desired.

APPLICATION CREATION TECHNIQUES

HS/FORTH assembles to any segment to create stand alone programs of any size. The optimizer can use HS/ FORTH as a macro library, or complex macros can be built as colon words. Full forward and reverse labeled branches and calls complement structured flow control. Complete syntax checking protects you. Assembler programming has never been so easy.

The Metacompiler produces threaded systems from a few hundred bytes, or Forth kernels from 2k bytes. With it, you can create any threading scheme or segmentation architecture to run on disk or ROM. You can turnkey or seal HS/FORTH for distribution, with no royalties for turnkeyed systems. Or convert for ROM in saved, sealed or turnkeyed form.

HS/FORTH includes three editors, or you can quickly shell to your favorite program editor. The resident full window editor lets you reuse former command lines and save to or restore from a file. It is both an indispensable development aid and a great user interface. The macro editor provides reusable functions, cut, paste, file merge and extract, session log, and RECOMPILE. Our full screen Forth editor edits files or sector mapped blocks.

Debug tools include memory stack dump, memory map, decompile, single step trace, and prompt options. Trace scope can be limited by depth or address.

HS/FORTH lacks a "modular" compilation environment. One motivation toward modular compilation is that, with conventional compilers, recompiling an entire application to change one subroutine is unbearably slow. HS/FORTH compiles at 20,000 lines per minute, faster than many languages link — let alone compile! The second motivation is linking to other languages. HS/FORTH links to foreign subroutines dynamically. HS/FORTH doesn't need the extra layer of files, or the programs needed to manage them. With HS/FORTH you have source code and the executable file. Period. "Development environments" are cute, and necessary for unnecessarily complicated languages. Simplicity is so much better.

HS/FORTH Programming Systems

Lower levels include all functions not named at a higher level. Some functions available separately.

- Documentation & Working Demo
  (3 books, 1000 + pages, 6 lbs) $ 95
- Student
  $145
- Personal optimizer, scaled & quad integer
  $245
- Professional 80x87, assembler, turnkey
  $395
- dynamic strings, multitasker
  $495
- RSDL linker
  $495
- physical screens
  $495
- Production ROM, Metacompiler, Metawindows
  $495
- Level upgrade, price difference plus
  $ 25
- OBU modules
  $495
- Rosetta Stone Dynamic Linker
  $ 95
- Metawindows by Metagraphics (includes RSDL)
  $145
- Hardware Floating Point & Complex
  $ 95
- Quad integer, software floating point
  $ 45
- Time slice and round robin multitaskers
  $ 75
- GigaForth (80286-386 Native mode extension) $295

HARVARD

SOFTWORKS

PO BOX 69
SPRINGBORO, OH 45066
(513) 748-0390
I don’t write Forth code for a living, so I have a little leeway and the luxury to look at the artistic side of Forthwriting. Yes, I feel that writing Forth code and its commentary can be an artistic endeavor. That’s not to say that the world of Forth commenting is crying out for its e.e. cummings or William Burroughs. But even technical manual writing, a well-known rite of commenters, and even a technical manual can be beautiful. Psychologists now affirm that each of us has a sense of beauty inside himself, and the current fashion allows each person free expression of that sense. Beauty and clarity are not antithetical. Let’s encourage commenting as a creative activity, no less laudable than the code writing itself.

Best!
Nathaniel Grossman
Department of Mathematics
UCLA
Los Angeles, California 90024

Outa’ Space
Dear Editor,

I am in need of help from my fellow FIGers. I have been working, on and off, on a program that was to produce printed circuit artwork with computer-driven routing, as well as other bells and whistles. The output is to an Epson-compatible printer or plotter. I hoped to release this program as an aid to the hardware experimenter.

My problem is that I have run into a space limitation. I am using F83 on a PC clone, and have reached the top of usable memory. I have utilized every space-saving technique I can think of, including reusing the disk buffers for a circular queue when necessary. When compiling, I use the assembler as a temporary module, and do not include the editor, but I still end up with a .COM file of >57K, not including the >150K of memory usage outside the Forth segment. The program is only partially operational, and at present consists of three text files totalling 4000 lines of code. There is still more to be added, but without more room it cannot be done.

The only solution to my problem seems to be an overlay manager for F83. Portions of the program are mutually exclusive, and would lend themselves to an overlay procedure.

If I bang my head against the wall long enough, I may be able to come up with something—but I would like to complete this project before the end of the decade. If anyone out there can help me, or is interested in the program, please drop me a line.

George Boudreau
P.O. Box 431
Kentville, Nova Scotia
Canada B4N 3X3

Korean Forth Is Natural
Sir:

I am enclosing my membership dues and photocopies of two articles my colleague, Jin-Mook Park, and I have written to introduce Korean Forth, which is a translated version of fig-FORTH. Our appreciation goes to the Forth inventor, Charles Moore, and all others who contributed to the development of Forth.

Since the syntax of Forth is similar to that of the Korean language, it is the appropriate computer language to be translated into Korean. Even though some of the other computer languages have been translated into Korean, they look strange to Koreans because of their English-like syntax. Thus, we translated fig-FORTH into Korean for Apple II+ compatibles, which have the capability to display lower-case letters. The character-generation ROM was modified to replace the lower-case patterns with those of 24 Korean characters.

There is a special notation for the Korean language. A syllable is expressed with 2–5 characters, and is normally written as a cluster character. Cluster characters are very difficult to display on the monitor, almost as difficult as displaying Chinese characters. Fortunately, Korean characters can be placed one by one as in Roman languages, with the characters in a queue. Although the latter writing method is not popular, it enabled us to implement Korean Forth with minimum modification to the hardware, i.e., replacement of the character-generation ROM.

Our first article describing Korean Forth was published by the Korean computer magazine *micro software* in September 1987. We described Forth briefly by comparing the syntaxes of computer languages like LOGO and Forth with that of the Korean language, and gave word-for-word translations of Forth words. The second, two-part, article was published in *Scientific Eastern Asian* in February and March 1988. The first part, “A Computer Understands the Korean Language” introduces the general features of Korean Forth, with printouts performed by LIST, VLIST, and INDEX. The second part, “Let’s Program in Korean,” is for the introduction of Forth programming. Screens containing definitions for a turtle graphics demonstration are listed and explained.

Forth is so simple and elegant that we, as an electronics technician and a chemist, could implement Korean Forth. Many thanks for the great works at FIG: publicizing and advancing the Forth language.

Sincerely,
Chong-Hong Pyun
Inorganic Chemistry Lab, KAIST
P.O. Box 131, DongDaeMun
Seoul, 130-600
Korea
**1989 ROCHESTER FORTH CONFERENCE**  
**ON**  
**INDUSTRIAL AUTOMATION**

*June 13 – 17, 1989*  
*University of Rochester*  
*Rochester, New York*

**CALL FOR PAPERS**

There is a call for papers on the following topics:

- Forth-based sensor systems
- Industrial security systems
- Process control
- Materials handling
- Robotics
- Tele-operated machinery

In addition, papers on all aspects of Forth application, implementation, and technology are being solicited. These include biomedical, avionics, and space-based, business and other applications; conventional processor implementations and Forth machines; and the continuing development of Forth tools and the ANSI X3J14 Forth Standard.

Papers may be presented in either platform or poster sessions. Please submit a 200 word abstract by May 15th. Papers should be received by June 1st, and are limited to a maximum of four single spaced, camera-ready pages. Longer papers may be presented at the Conference but should be submitted to the refereed Journal of Forth Application and Research. Abstracts and papers should be sent to the Conference Chairman: Lawrence P. Forsley, Instituto Applied Forth Research, Inc., 70 Elmwood Avenue, Rochester, NY14611, or, electronically to LForsley on BIX or LFORSLEY on GENie.

For more information please contact the Conference Chairman or call (716)-235-0168.

---

**REGISTRATION**

The registration fee and conference services includes all sessions, meals, and the Conference papers. Lodging is available at local motels or in the UR dormitories.

Registration will be from 4-11 PM on Tuesday, June 13th in the Wilson Commons, and from 8 AM Wednesday, June 14th in Hutchison Hall, where sessions will be held.

<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
<th>Telephone: Wk (<strong><strong><strong>) Hm (</strong></strong></strong>)</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Registration:</th>
</tr>
</thead>
<tbody>
<tr>
<td>$200.00</td>
</tr>
<tr>
<td>$50.00 (full-time students)</td>
</tr>
<tr>
<td>Vegetarian Meal Option</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Conference Services</th>
<th>Dormitory housing, 5 nights</th>
</tr>
</thead>
<tbody>
<tr>
<td>$200.00</td>
<td></td>
</tr>
</tbody>
</table>

- $125.00 single  
- $100.00 double  
- 5K Fun Run  
- non-smoking roommate

Total $_______  
Amount Enclosed $_______

MC/Visa# _______ exp _______  

Please make checks payable to the Rochester FORTH Conference. Mail your registration to Rochester FORTH Conference, Box A, 70 Elmwood Avenue, Rochester, NY 14611, USA.
There is no agreement about what constitutes a good form for source code in any language. Programming languages are too cryptic for easy understanding by the English reader. Some form of documentation is often included that, in one way or another, tries to help the reader. Having examined a variety of styles used by a number of programming languages, I have experimented with several for use with Forth programs.

In the Forth community, source code was first written in a series of blocks with occasional comments enclosed in parentheses. Further comments were sometimes included in shadow screens. These blocks or screens corresponded to the size of the screen Charles Moore had available when he first created Forth. Recently, Forth source code has been written as text files but it still can only be viewed in pages, according to the device used. Even in scrolled text, the displayed window into the text is a sort of a page. We are always back to pages.

In many journals, source code is presented in some way separate from the narrative text. The code may be inserted between lines of text as examples or included in an appendix. If one assumes that the publication of source code requires some sort of narrative text, perhaps one should simply write all of the source code as a narrative.

Conventional source code for most languages uses some sort of enclosing symbols for comments. This technique would seem to have things backwards. What is needed is some sort of enclosing symbols to indicate what part of the narrative should be compiled. The rest of the narrative is simply easily readable documentation. (This presumes that the programmer can write English, which may not always be the case.)

With the simplicity of Forth, I have decided to see what such a technique would be like. I have selected a piece of code I wrote some time ago and published in the "FORML Proceedings 1981. That code was written in fig-FORTH. It really needed updating to the code I presently use and seems to be a good, short piece to use as an example.

```
: FILE-COMPIL
TIB @ >IN @
 1 >IN +!
C@ 123 =
UNTIL ;
IMMEDIATE
```

```
IMMEDIATE
: FILE-COMPIL
TIB @ >R >IN @ >R
FILE-BUFFER TIB !
0 BLK ! 0 >IN !
[COMPIL] ) <INTERPRET>
R> >IN ! R> TIB ! ;
```

One could easily eliminate all the documentation by rewriting the source file to include only the parts enclosed within braces. This can be done with a simple filter function, which can be easily implemented in Forth. But this would defeat the purpose of including documentation.

If one develops his code as a text file, he could easily edit his code to conform with a style. And, really, the best time to do that is while the code is fresh in his head. In addition to an introductory note and an overview of the code, there are as many ways of bringing a text file into a Forth program as there are programs. I will leave that to the option of the user.

The enclosing symbols are the opening and closing braces. The opening brace is simply a text marker and requires no definition. The closing brace is an immediate Forth word to search for the next opening brace from which to start compiling again. The change required in INTERPRET is to point the interpreter to the text buffer in memory and to start interpreting at the first open brace.

"All the information is easily located where it belongs."

The source code can be easily compiled in MVP-FORTH by a routine to bring a text file into a buffer and a slight modification to INTERPRET. With some slight modifications to FPC version 2.5, the same source code has compiled successfully. There are as many ways of bringing a text file into a Forth program as there are programs. I will leave that to the option of the user.

The enclosing symbols are the opening and closing braces. The opening brace is simply a text marker and requires no definition. The closing brace is an immediate Forth word to search for the next opening brace from which to start compiling again. The change required in INTERPRET is to point the interpreter to the text buffer in memory and to start interpreting at the first open brace.
The discipline of completing the programming job by adopting such a style would meet many common problems with understanding and maintaining code. All of the necessary information is easily located where it belongs. Also, publication is the ultimate goal of much research; why not incorporate that style in the source code and have the job done?

The serial-day-compression source code which follows is presented as an example for consideration. In the end, the style adopted is left up to the programmer and his management. Any style could be used. The important thing is that some style be adopted which serves as many functions as possible. This source code exists and works. It is not vaporware—try it!

### SERIAL DAY COMPRESSION

by Glen B. Haydon

A serial day can be converted to a 16-bit value. The converted value can conserve memory space and make calculation of the period between days easy. The following algorithm is adapted from one presented in Sky & Telescope (April 1981, page 312), and was published in FORML Proceedings 1981.

The military sometimes refers to a Julian Day when they really mean a serial year day. It would take more than 16-bits to encode a Julian Day—in that form of reckoning, we are nearly up to 2,300,000 and each date includes an additional fractional part to indicate the time. Julian days also start at noon. This program provides an offset from the Julian Day, with March 1, 1952 being day 123. By beginning with March first, the leap-year day is the immediately preceding day. The time fraction is also dropped and the days are calendar days. The fractions in the original algorithm are used to calculate leap years and the variable lengths of months. They have been scaled to integer values and then truncated to give the proper values for a 16-bit Forth system.

The source is divided into four parts:

1. Two mathematical operators in addition to the program kernel are desirable.
2. Encoding a 16-bit serial day is accomplished with ?DATE which first prompts for the input in a prescribed format, parses the input to three double-precision values with S~N and then these values are scaled and combined with the single function TO.SERIAL.DAY.
3. Decoding a 16-bit serial day is more complicated because it is necessary to determine if the year is a leap year and, accordingly, make a number of adjustments. This part leaves three values on the stack ready to be formatted for output.
4. Finally, the three decoded values can be formatted in a variety of ways of which one example is given.

### Additional Mathematical Operators

D/
A double-precision number is divided by a single-precision unsigned number and leaves a double-precision value.

#### Implementation

```forth
: D/ ( d u -- d )
  SWAP OVER /MOD >R SWAP
  U/MOD SWAP DROP R> ;
```

### Test

4. 2 D/ D.
Will print a double-precision value of 2.

### Comment
The function illustrates how to write uncommon specific code for an application.

D*
A double-precision number is multiplied by a single-precision unsigned number to leave a double-precision value.

#### Implementation

```forth
{ : D* ( d u -- d )
  DUP ROT * ROT ROT U* ROT + ;
}
```

#### Test

4. 2 D* D.
Will print the double-precision value 8.

### Comment
The result could overflow if the original values are too large. This is not a problem in this application, so no error checking is done.

### Encoding a Serial Day

TO.SERIAL.DAY
Convert a series of double-precision values representing the month, the day, and the year in the form MM/DD/YY to a 16-bit serial day.

#### Implementation

```forth
: TO.SERIAL.DAY ( d d d -- u )
  ROT DUP 3 < IF 13 + SWAP 1 -
  ELSE 1+ SWAP THEN
  52 - 365.25 ROT D* 100 D/ DROP
  SWAP 30.6001 ROT D* 10000 D/ DROP + + ;
```

---

Volume X, Number 6
Test
3. 30. 60. TO.SERIAL.DAY
Should leave a single 16-bit value.

Comment
This is the heart of the encoding. No error checking is done. The result can be checked after input and reentered if necessary. The value 13 is introduced to offset the months in the calculation to begin with March.

$-N
Parse the next character string according to the delimiter provided, and convert it into a double-precision value.

Implementation
{
  : $-N  ( c -- d )
    WORD 0 0 ROT CONVERT DDROP ;
}

Test
: TEST
  ." Enter numerical value-->
  PAD 10 EXPECT BL $-N D. ;
Should print the value entered.

Comment
The function is a factor in parsing a date input formatted with slashes between the values and terminating with a blank.

?DATE
Provide a prompt to remind the user of the format for the input, and convert the input to a serial day.

Implementation
{
  : ?DATE
    " (MM/DD/YY) --> "
    QUERY 47 $-N 47 $-N
    BL $-N TO.SERIAL.DAY ;
}

Test
?DATE
Will issue the desired prompt and leave a 16-bit serial day, which can be printed with .DATE to be coded later.

Comment
These several routines have no error checking and some means should be provided to ensure that the intended date has been entered. There is no possible way to assure that the intended date has been input, so one might as well inspect the result and eliminate the overhead for error checking.

Decoding Routines
YEARS
From a serial day value, calculate a test year as a single-precision integer from which it can be determined if a leap year must be considered.
Implementation
{
 : YEARS ( serial-day -- test-year )
  0 100 D* 36525 D/ DROP ;
}

Test
123 YEARS
Should leave the value 0.

Comment
Making the value a double-precision integer is needed for scaling.
The truncation takes care of the leap year.

DAYS/YEARS
For a given year, calculate the number of days in that year.

Implementation
{
 : DAYS/YEARS ( year -- days )
  0 36525 D* 100 D/ DROP ;
}

Test
123 YEARS DAYS/YEARS
Should leave the number of days in the selected year.

Comment
The scaling avoids the need for floating point.

TEST.YEARS
From the serial day and the test year calculated above, determine the
actual year and the number of remaining days.

Implementation
{
 : TEST.YEARS ( serial-day test-year -- year days )
  DDUP DAYS/YEARS - DUP 123 <
    IF DROP 1- SWAP OVER DAYS/YEARS -
    ELSE ROT DROP
    THEN SWAP 52 + SWAP ;
}

Test
123 0 TEST.YEARS
Should convert the serial day to the year 52 and the remaining days
in that year.

Comment
The number of years determined is offset to begin with 52, which
in this case is the year the sample date should give.

MONTHS
From the remaining days after the test year has been removed,
determine a test month.

Implementation
{
 : MONTHS ( days -- days test-month )
  DUP 3267963. ROT D*
  10000 D/ 10000 D/ DROP ;
}

Test
123 MONTHS .S
Should show the days and a value for a test-month.

Comment
The rather large value for determining a test month must be scaled
down in two steps on a 16-bit processor.

DAYS.TO.M/D/Y
Going back to the calculation leaving years and days, calculate the
actual number of months and days.

Implementation
{
 : DAYS.TO.M/D/Y ( years days -- years days months )
  MONTHS SWAP OVER
  30.6001 ROT D* 10000 D/ DROP - SWAP DUP 13 >
    IF 13 - ROT 1+ ROT ROT
    ELSE 1- THEN ;
}

Test
52 123 DAYS.TO.M/D/Y .S
The three values should be left on the stack ready to be output.

Comment
Again the use of double-precision integers with scaling is used
rather than floating point. Note the 13 is used to gain the offset to
the effective year ending at the end of February.

CONV.SERIAL
Convert a serial day to a form ready for formatting.

Implementation
{
 : CONV.SERIAL ( serial-day -- years days months )
  DUP YEARS TEST.YEARS DAYS.TO.M/D/Y ;
}
Test
123 CONV.SERIAL
Should complete the conversion from the serial day to 3/1/52.
Try other values.

Comment
With the values left on the stack, they may be used in many ways, according to the requirements of the application.

Format and Output
OUT.DATE
Format the date values into a double-precision value, which can then be formatted with the primitive formatting tools, and finally type it.

Implementation
{
: OUT.DATE ( years days months -- )
  100 * + 0 100 D* ROT 0 D+
  # # # 47 HOLD # # 47 HOLD # # #>
  TYPE ;
}

Test
52 1 3 OUT.DATE
Should output the beginning date in the form 3/1/52.

Comment
A variety of other output formats could be implemented, according to the desires of the user.

.DATE
From a serial day as it may be retrieved from a database, print the date in the selected format.

Implementation
{
: .DATE ( serial-day -- )
  ?DUP
  IF CONV.SERIAL OUT.DATE
  ELSE ." 00/00/00" THEN ; EXIT
}

Test
123 .DATE
Should print the date 3/1/52.

Comment
The program can be rearranged to output the result in any different format, such as DD/MM/YY. If the serial day has a zero value, as in an unused zero-initialized date field, the zero date is printed without any conversion.

---

SDS FORTH for the INTEL 8051

Cut your development time with your PC using SDS Forth based environment.

Programming Environment
- Use your IBM PC compatible as terminal and disk server
- Trace debugger
- Full screen editor

Software Features
- Supports Intel 805x, 80C51FA, N80C451, Siemens 80535, Dallas 5000
- Forth-83 standard compatibility
- Built-in assembler
- Generates headerless, self starting ROM-based applications
- RAM-less target or separate data and program memory space

SDS Technical Support
- 100+ pages reference manual, hot line, 8051 board available now

Limited development system, including PC software and 8051 compiled software with manual, for $100.00. (generates ROMable applications on top of the development system)

SDS Inc., 2865 Kent Avenue #401, Montreal, QC, Canada  H3S 1M8  (514) 461-2332

---

Forth Dimensions 14  Volume X, Number 6
A HIGH-LEVEL SINGLE-STEPPER

PHILIP BACON - GAINESVILLE, FLORIDA

When a newly defined Forth word fails to produce the intended effects, the source of the difficulty can sometimes be found by a single-stepper, a program that executes the words of a colon definition one-by-one, displaying the contents of the stack and pausing at each step until a keyboard command causes it to continue.

When I switched from programming a Commodore 64 to an IBM XT, I wanted a single-stepper immediately, without waiting to learn 8088 assembly language. So I designed the single-stepper shown in screens 23–29. The basic strategy is to simulate the Forth interpreter. A variable IP serves as a mock instruction pointer. An array RETURNSTACK is used to simulate the return stack; the parameter stack simulates itself. The only Forth words that do not simulate themselves are those that manipulate the instruction pointer or the return stack in unusual ways.

Dissect Your Interpreter

Since different Forth interpreters behave differently, this is not a program you can just type in and run. To write a single-stepper that works for you, you will have to find out just how your Forth interpreter behaves. One way to do this is to read the source code for your Forth. An easier alternative is to run a few test words and observe the results. Here is an example of the kind of detective work to be done.

The Forth-83 words that interact with the return stack are >R, R>, R@, QUIT, ABORT, ABORT", EXIT, and the run-time actions of ;, ;, and ;CODE. Additional possibilities are the run-time words for DO, LEAVE, LOOP, and +LOOP. An examination of the word list for the Forth at hand turns up yet more possibilities: 2>R, 2R>, S, R0, RP0, RP!, and UNRAVEL.

To this list of words that might need to be simulated, we must append words that use the instruction pointer in special ways. These include the words that push compiled constants to the parameter stack, and the run-time words compiled by IF, ELSE, THEN, BEGIN, WHILE, REPEAT, AGAIN, and ;. In the Forth at hand, these run-time words are LIT, DLIT, 0BRANCH, BRANCH, and (;). Any words in the list that are defined in terms of other words in the list do not need to be simulated. We now start probing.

The word SCAN (screen 21) displays the compiled form of colon definitions. In response to

: ALPHA DUP ;
SCAN ALPHA

the following is displayed:

7B25 D9 05 19 05 CB 03
NGS FORTH
A FAST FORTH, OPTIMIZED FOR THE IBM PERSONAL COMPUTER AND MS-DOS COMPATIBLES.

STANDARD FEATURES INCLUDE:

- 79 STANDARD
- DIRECT I/O ACCESS
- FULL ACCESS TO MS-DOS FILES AND FUNCTIONS
- ENVIRONMENT SAVE & LOAD
- MULTI-SEGMENTED FOR LARGE APPLICATIONS
- EXTENDED ADDRESSING
- MEMORY ALLOCATION CONFIGURABLE ON-LINE
- AUTO LOAD SCREEN BOOT
- LINE & SCREEN EDITORS
- DECOMPILER AND DEBUGGING AIDS
- 8088 ASSEMBLER
- GRAPHICS & SOUND
- NGS ENHANCEMENTS
- DETAILED MANUAL
- INEXPENSIVE UPGRADES
- NGS USER NEWSLETTER

A COMPLETE FORTH DEVELOPMENT SYSTEM.

PRICES START AT $70

NEW HP-150 & HP-110 VERSIONS AVAILABLE

NEXT GENERATION SYSTEMS
P.O.BOX 2987
SANTA CLARA, CA. 95055
(408) 241-5909

SCR # 23
0 (STEP )
1 DECIMAL
2 VARIABLE IP
3 : IP+ 2 IP +
4 : SKIP 4 IP +
5 VARIABLE LEVEL
6 VARIABLE MARK
7 VARIABLE RETPTR
8 : RP@ RETPTR @
9 : RP! RETPTR !
10 VARIABLE :RO
11 CREATE RETURNSTACK 64 ALLOT
12 HERE :RO !
13 -->
14
15

SCR # 24
0 (STEP )
1 DECIMAL
2 :R@ RETPTR @ @ ;
3 :R> :R 2 RETPTR + ! ;
4 : :R -2 RETPTR + ! RETPTR @ ! ;
5 : :S IP @ DUP 2+ :R @ 2+ IP ! 2 LEVEL + ! ;
6 : :S :R IP ! -2 LEVEL + ! ;
7 : PRIMARY IP @ @ IP+ EXECUTE ;
8 : BRANCH IP @ 2+ @ IP + ;
9 : BRANCH IF SKIP ELSE :BRANCH THEN ;
10 : LIT IP @ 2+ @ SKIP ;
11 : DLIT IP @ 2+ DUP @ SWAP 2+ @ 6 IP + ! ;
12 : 2>R SWAP :R :R ;
13 : 2R> :R> SWAP ;
14 -->
15

SCR # 25
0 (STEP )
1 HEX
2 : :DO)
3 : IP @ 2+ @ :R OVER 8000 + :R OVER 8000 SWAP - :R SKIP ;
4 : :LOOP ) :R> 1+ DUP 8000 - IF
5 : DROP :R> DROP :R> DROP SKIP
6 ELSE :R> BRANCH THEN ;
7 : :(+LOOP ) :R> OVER OVER + DUP :R> 0<
8 IF 0< NOT SWAP 0< NOT ELSE 0< SWAP 0< THEN AND
9 IF :R> DROP :R> DROP :R> DROP DROP SKIP ELSE :BRANCH THEN ;
10 : LEAVE :R> DROP :R> DROP :R> IP ! ;
11 : I :R> DUP :R@ + SWAP :R> ;
12 : J RETPTR @ DUP 6 + @ SWAP 8 + @ + ;
13 : :(CODE) :R> LATEST NAME> ! -2 LEVEL + ! ;
14 ' DUMP @ CONSTANT DOCOL
15 DECIMAL -->

Forth Dimensions 16 Volume X, Number 6
SCR # 26
0 (STEP)
1   : PACE IP @ @
2   |' R@ =IF IP+ :R@ FIN
3   |' R+ =IF IP+ :R+ FIN
4   |' >R =IF IP+ :R FIN
5   |' i :S =IF :S FIN
6   |' EXIT =IF :S FIN
7   |' BRANCH =IF :BRANCH FIN
8   |' OBRANCH =IF :OBRANCH FIN
9   |' (DO) =IF :DO FIN
10  |' (LOOP) =IF :LOOP FIN
11  |' (+LOOP) =IF :(+LOOP) FIN
12  |' LEAVE =IF :LEAVE FIN
13  |' I =IF IP+ :I FIN
14  |' J =IF IP+ :J FIN
15  |' (;CODE) =IF ;(CODE) FIN -->

SCR # 27
0 (STEP)
1   |' LIT =IF :LIT FIN
2   |' DLIT =IF :LIT FIN
3   |' 2>R =IF :2>R FIN
4   |' 2R =IF :2R FIN
5   |' R0 =IF :RO FIN
6   |' RP@ =IF :RP@ FIN
7   |' RP! =IF :RP! FIN
8   |' QUIT =IF QUIT FIN
9   | DOCOL - IF :: ELSE PRIMARY THEN ;
10  -->
11
12
13
14
15

SCR # 28
0 (STEP)
1   HEX
2   CR CR 0 OUT ! ;
3   HTAB { n -- }
4   DUP OUT @ < IF CR ELSE OUT @ - THEN SPACES ;
5   U.S DEPTH IF 0 DEPTH 2- DO I PICK U. -1 -LOOP THEN ;
6   DISPLAY 18 HTAB " " U.S CR
7   IP @ DUP ADDRESS LEVEL @ SPACES @ >NAME ID . ;
8   SELECT
9   -1 SWAP
10  51 ( Q ) =IF DROP QUIT FIN
11  53 ( S ) =IF LEVEL @ 2+ MARK ! FIN
12  52 ( R ) =IF LEVEL @ MARK ! FIN
13  43 ( C ) =IF LEVEL @ 2- MARK ! FIN
14  DROP DROP 0 ; DECIMAL -->
15

SCR # 29
0 (STEP)
1   DECIMAL
2   VARIABLE ACTION
3   :STEP
4   |' ACTION ! ACTION IP ! 0 LEVEL !
5   ;R0 @ :RP! BEGIN
6   PACE [ ACTION 2+ ] LITERAL IP @ -
7   WHILE
8   LEVEL @ MARK @ > NOT IF
9   DISPLAY BEGIN KEY SELECT UNTIL THEN
10  REPEAT ;
11
12
13
14 ;S
15
By using we find that the compilation address of DUP is 0519 (hex), and the compilation address of ;S is 03CB. Comparison of these values with the compiled form of ALPHA suggests that compilation addresses are compiled in low-high byte order, that every colon definition is prefixed by the bytes D9 and 05, and that ;S compiles ;S.

The next test requires two words:

: BETA R> DUP
 .ADDRESS >R ;

: GAMMA BETA ;

The output of SCAN GAMMA is: 7B46 D9 05 32 7B CB 03

The compilation address of BETA is 7B32. Execution of GAMMA prints 7B4A. This shows that when the interpreter encounters a colon definition, it adds two to the value of the instruction pointer before pushing the value to the return stack. Our simulator must treat IP and RETURN-STACK similarly.

The next word is used to find out how branches are compiled.

: DELTA
 IF DUP ELSE SWAP THEN ;

The output of SCAN DELTA is:
7B5E D9 05 8A 01 60 7B 19 05
7B5C 77 01 62 7B 08 05 CB 03

The compilation addresses of BRANCH and OBRANCH are 0177 and 018A, respectively. Knowing this, we can see that IF compiles OBRANCH followed by a destination address, and that ELSE compiles BRANCH followed by a destination address. Our simulation of OBRANCH must put the compiled address into IP if the top of the parameter stack is zero; otherwise, the value of IP must be incremented by four. The simulation of BRANCH must put the compiled address into IP.

The next test word exhibits the compiled form of a DO loop.

: EPSILON
 DO LEAVE LOOP ;

The output of SCAN EPSILON is:
7B6E D9 05 E0 01 7A 7B DF 03
7B7A A8 01 74 7B CB 03

The compilation address of (DO) is 01E0, that of LEAVE is 03DF, and that of (LOOP) is 01A8. We see that (DO) is followed by the address of the loop exit, and (LOOP) is followed by the address of the first location within the loop.

Execution of the next test word shows that (DO) pushes the DO loop parameters to the return stack.

: OMEGA
 5 2 DO
  CR I .
  R> DUP .ADDRESS
  R> DUP .ADDRESS
  R> DUP .ADDRESS
  >R >R >R
  LOOP ;

Execution of OMEGA yields:
2 7FFD 8005 7BB2
3 7FFE 8005 7BB2
4 7FFF 8005 7BB2

At run time, (DO) pushes three items to the return stack: the address of the loop exit, the sum of 8000 and five, and the value 8000 - (5 - 2). Each time (LOOP) is encountered, the value on the top of the return stack is incremented by one. When the result is 8000 (hex), the loop is exited. Execution of I produces the sum of the first two values on the top of the return stack. When LEAVE executes, control goes to the third address from the top. Our simulations of these words must treat RETURNSTACK similarly.

If FIVE is defined as:

: FIVE 5 ;

execution of SCAN FIVE yields
7B24 D9 05 42 01 05 00 CB 03

Since 0142 is the compilation address of LIT, we see that the simulation of LIT must push the value contained in the two bytes following LIT, and increment that value of IP by four.

You might imagine that ("" which is the run-time word for ")"— would need to be treated something like
LIT. But in this version of Forth, ( . " ) is a compound word, and therefore can simulate itself.

Using the Single-Stepper

Once we have simulations for the words that need them, we put all in a word called PACE. Suppose IP contains an address at which the compiling address of a Forth word resides. If PACE is invoked and the Forth word is not a compound word, that word is executed. If the word is defined by a colon definition, the address of the first word in the colon definition is put into IP and the value of a variable LEVEL is the principal building block for the single-stepper STEP, which is used as follows.

Suppose you have defined a Forth word SHAKY that doesn't work as you intend. Put any parameters needed by SHAKY onto the parameter stack and type STEP SHAKY. The contents of the parameter stack and the first word in the definition of SHAKY will be displayed. If you now press the R key, the displayed word will be executed, a new picture of the stack will be shown, and the next word to be executed will be displayed. Press the R key repeatedly, until the word that produces the bad output (TROUBLE, say) is executed. Quit (by pressing Q) and start over. But this time, when TROUBLE is displayed, do not execute it by pressing R; instead, press S. If TROUBLE is a compound word, the first word in its definition is displayed. The descent into the definition of TROUBLE is indicated by extra indentation. To further localize the source of your difficulties, step through the definition of TROUBLE by repeatedly pressing R. To complete the execution of whatever word you are currently stepping through, press C.

Philip Bacon teaches mathematics at the University of Florida. He has used a homebrew Forth to prototype assembly language programs for the Commodore 64.
After studying several books about the Forth programming language, I decided the only way to really learn Forth would be to attempt to write a program using it. About this time, my eight-year-old son wanted me to help him write a program to draw on the screen. We first wrote a BASIC program similar to the familiar Etch-a-Sketch, then I decided to translate it into Forth. This was easier than I thought, so I decided to write a more difficult program.

I took a BASIC program and rewrote it to run with Laxen & Perry’s F83. This is a public-domain implementation, available on many bulletin boards. I got my copy from PC-SIG (1030 E. Duane, Suite J, Sunnyvale, CA 94086; the two-disk set 263 and 264).

In this game, you may choose to play against up to three opponents, deadly beasts which stalk you as you try to trap them while staying out of their reach. This version of the game is neither an elegant nor an efficient translation into Forth, but it is faster than the BASIC version. I had to put a timing loop into the Forth code to slow the game down. I built in nine different speeds (not in the original game), as well as three levels of difficulty.

When the game is played, a title page shows which keys move your piece around the screen. Use the keypad, as shown below, to move. Pressing 8 moves your screen marker up, 1 moves it diagonally down to the left, etc. Press any other key to stop moving.

Obstacles are placed randomly on the screen. You can push these around, but the beasts cannot. Your job is to build walls completely enclosing the beasts. The game ends when none of the beasts can move (or if you get eaten).

**Programming Notes**

Obstacles, beasts, and your marker are placed randomly on the text screen. The program pauses to allow the player to study the game board and plan a strategy.

In the highest level of difficulty, each beast gets a chance to move every time you do. At lower levels you are given extra moves.

"I had to put in a timing loop to slow the game."

Screens one through five contain general-function words. The game itself begins on screen six. Screens seven and eight display the title and get the required input to play the game. Screen nine sets up a white-on-blue game board and outlines the board with double box characters.

Screens 12-16 contain the words that move the marker around the board. MOVE_BEAST (screen 18) contains the routine for moving the beasts. The beasts follow you, but not perfectly. They have to be able to move around obstacles, even if that entails backing up. After checking to see if the beast is right next to you (too bad for you!), a value of -1, 0, or 1 is selected for X and Y. This will modify the beast’s position coordinates so that there is a higher probability of it stepping toward you than away from you. A check is made in the chosen location for an ASCII 32 (space). If an obstacle or another beast is found there, the beast searches in adjacent squares for a favorable place to move. When no beasts can move, you have won the game. LOSE? and WIN? (screen 19) check for these conditions and take the appropriate action.

Code on screen 23 will create a CAPTURE.COM file. This eliminates the need to enter Forth and load the program each time you wish to play.

During the game, the elapsed time is displayed.

**Credits and Requirements**

Capture is designed to run on black-and-white or color monitors in 80-column mode. This version was written for the IBM PC.

The original version of this game was called Trap’Em, written by Rob Smythe in Applesoft BASIC for the Apple II computer, and was published in *Nibble* (vol. 2, no. 4, 1981).
This game is based on the program originally written in Applesoft BASIC by Rob Saythe for the Apple II and published in NIBBLE magazine Vol.2/No.4/1981. I wrote this version of the program as a learning experience. It is written using the public domain version of FORTH - F33 by Perry Laxen. I wrote this version of the program as a learning experience. It is written using the public domain version of FORTH - F33 by Perry Laxen.

Bruce T. Nicholas

Variables used by timer.

Variables used by timer.

GETIME AX FOP AL AN MOV 33 INT CX PUSH DX PUSH CODE GETIME Get the time of day.

AH AH SUB AL PUSH NEXT END-CODE

TIMEA 44 GETIME DROP 256/ROT 256/ROT ; : TIMEA Get the time.

STORE the time in Hours, Minutes and seconds.

TIME TIMEA HOURS ! MINUTES ! SECONDS ! DROP ; : TIME! Get the time and format into minutes and seconds to show elapsed time during game.
3 0 \ CODE WORDS - POKE, PEEK, EQUIP, CGA? BTN 04-20-1988 \ CODE WORDS - POKE, PEEK, EQUIP, CGA? BTN 04-14-1988
2 CODE POKE (S byte segment address -- ) CODE POKE Used to put a byte of data to the Video screen.
3 BX POP ES POP AX POP ES: AL 0 [BX] MOV NEXT END-CODE
4 CODE PEEK (S segment address -- byte ) CODE PEEK Used to get a byte of data from the screen.
5 AX AX SUB BX POP ES POP ES: 0 [BX] AL MOV 1PUSH END-CODE
6 CODE EQUIP (S -- equip ) CODE EQUIP Puts a list of hardware on the stack. Used to
determine the type of attached monitor.
7 17 INT 1PUSH END-CODE \ list of hardware on stack

9 HEX
10 : CGA? (S -- video-buffer-address attrib ) \ B&W or Color : CGA? Used to set the memory location of the attached
11 EQUIP 30 AND 31 = IF OB000 ( B/W MONITOR) 70 (ATTRIB) then ;
12 ELSE OB000 ( COLOR MONITOR) 71 (ATTRIB) then ;
13 DECIMAL
14 : CGA? CONSTANT ATTRIB
15 CONSTANT SEGMENT

4 0 \ RANDOM GENERATOR - RAND, RANDOM, EKEY, WKEY BTN 04-20-1988 \ RANDOM GENERATOR - RAND, RANDOM, EKEY, WKEY BTN 04-18-1988
1 VARIABLE SEED TIMES + * # SEED ! VARIABLE SEED Used as a seed for the random generator.
2 : RAND (S -- random no. ) : RAND Used for the random generator.
3 SEED # 5421 # 1+ DUP SEED ! ;
4 : RANDOM (S n -- random no. ) : RANDOM Random number generator.
5 RAND FLIP SWAP MOD ;
6 10 : EKEY KEY DUP 0= IF DROP KEY THEN ; \ Extended keys : EKEY Reads the extended keyboard keys.
7 : WKEY EKEY 0= IF DROP THEN ; : WKEY Wait for any key press.
8
15

5 0 \ MATRIX WORDS - MATRIX, ELEMENT, FIND-LENGTH etc BTN 04-14-1988 \ MATRIX WORDS - MATRIX, ELEMENT, FIND-LENGTH etc BTN 04-18-1988
2 : MATRIX (S 2 bytes per entry ) : MATRIX Create a two dimensional array.
3 CREATE (S #rows #columns -- )
4 ZUP , , \ Remember the dimensions
5 # 2* ALLOT ;
6 : ELEMEN (S row# col# "matrix -- "element ) : ELEMENT Locate any data in the matrix.
7 DUP \R \R
8 ROT 2* \ SWAP 2* +
9 R> 4 + + ;
10 11 24 80 MATRIX SCREEN 24 60 MATRIX SCREEN Build data image of the screen.
12 : FIND-LENGTH (S address -- length ) : FIND-LENGTH Find the length of the data matrix.
13 SCREEN DUP 2* \ SWAP 2* ;
14 : FILL-ARRAY (S array-index -- ) : FILL-ARRAY Fill the matrix with data.
15 0 DO I 2* 0 I SCREEN ELEMENT ! LOOP ;
Variables used in the program and their initial value.

Variables

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Initial Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>XKEY</td>
<td>INT</td>
<td>0</td>
</tr>
<tr>
<td>Y</td>
<td>INT</td>
<td>0</td>
</tr>
<tr>
<td>X</td>
<td>INT</td>
<td>0</td>
</tr>
<tr>
<td>WIN</td>
<td>BOOL</td>
<td>false</td>
</tr>
<tr>
<td>CNT</td>
<td>INT</td>
<td>0</td>
</tr>
<tr>
<td>DIFFCNT</td>
<td>INT</td>
<td>0</td>
</tr>
<tr>
<td>ANOTHER</td>
<td>BOOL</td>
<td>false</td>
</tr>
<tr>
<td>BEASTS</td>
<td>INT</td>
<td>0</td>
</tr>
</tbody>
</table>

DISPLAY THE TITLE

Display the title page on the screen.

SHOW THE CURSOR CONTROL KEYS

Press any KEY to continue.

Pause the display so the screen can be read.

Read the key and limit the input.

Store the number of beasts to be used.

Store the difficulty level to be used.

Store the speed of the game.
0 \ CAPTURE - BORDERS and COLOR
   BTN 04-14-1988 \ CAPTURE - BORDERS and COLOR
1
2 : THERE (S h v -- )
3   SWAP SEGMENT - ROT SCREEN ELEMENT 2 ;
4
5 : T&B 79 1 DO 205 I 0 THERE POKE
6      205 I 23 THERE POKE LOOP ;
7 : SIDES 23 1 DO 186 0 I THERE POKE
8      186 79 I THERE POKE LOOP ;
9 : CORNERS 201 0 0 THERE POKE
10    200 0 23 THERE POKE
11    187 79 0 THERE POKE
12    188 79 23 THERE POKE ;
13
14 : BORDERS T&B SIDES CORNERS ;
15 : COLOR 3840 1 DO 23 SEGMENT 1 POKE 2 + LOOP ;

10 \ CAPTURE - OBSTACLES, BEAST
   BTN 04-14-1988 \ CAPTURE - OBSTACLES, BEAST
11
2 : HORIZ 78 RANDOM 1+ ;
3 : VERT 22 RANDOM 1+ ;
4 : H B H CNT @ 2* + ; ( horiz index to each beast ) : H B
   Horizontal index to each beast.
5 : V B V CNT @ 2* + ; ( vert index to each beast ) : V B
   Vertical index to each beast.
6
7 : OBSTACLES 400 0 DO 177 HORIZ VERT THERE POKE LOOP ;
8 : VACANT? BEGIN HORIZ DUP H B ! VERT DUP V B ! THERE PEEK
    BL = UNIL ;
9 : BEAST -1 CNT ! BEGIN 1 CNT +! VACANT?
10
11 : CNT A #BEASTS A 1 - = UNIL ;

11 \ CAPTURE - HERO and Misc
   BTN 04-14-1988 \ CAPTURE - HERO and Misc
12
2 : V+1 V B + + 1+ ; \ Index 4 2# : V+1
3 : V-1 V B + + 1- ; : V-1
4 : H+1 H B + + 1+ ; : H+1
5 : H-1 H B + + 1- ; : H-1
6 : V0 V B + + ; : V0
7 : H0 H B + + ; : H0
8 : H1 H B + + ; : H1
9 : V1 V B + + ; : V1
10
11 : OPOKE H0 V0 THERE POKE ;
12
13 : HERO 4 CNT ! VACANT?
14 : BL OPOKE
15 : OPOKE LOOP ;

\ Forth Dimensions 24 Volume X, Number 6
0 \ capture - up\^, down  BTN 04-14-1988 \ capture - up\^, down  BTN 04-14-1988
1 2 : vacant dup bl = ; 0 do  up\^ vo do  up\^  is location blank?
4 3 0 do  h0 v-1 i - there peek vacant if
5 4 opoke 177 h0 v-1 i - there poke
6 5 2 h0 v-1 i - there poke
7 6 v-1 vi ! leave else
8 7 42 = if leave then then loop ;
9 8 10 do  down 23 vo - 0 do  down  move one space down.
11 10 11 h0 v+1 i + there peek vacant if
12 11 opoke 177 h0 v+1 i + there poke
13 12 2 h0 v+1 i + there poke
14 13 v+1 vi ! leave else
15 14 42 = if leave then then loop ;
13 0 \ capture - left, right  BTN 04-14-1988 \ capture - left, right  BTN 04-14-1988
1 2 : left h0 0 do  left  move one space left.
3 12 3 h-1 i - vo  there peek vacant if
4 13 4 opoke 177 h-1 i - vo  there poke
5 14 5 2 h-1 vo  there poke
6 15 6 h-1 h1 ! leave else
7 16 7 42 = if leave then then loop ;
8 17 9 : right 79 h0 - 0 do  right  move one space right.
10 18 9 10 h+1 i + vo  there peek vacant if
11 11 11 opoke 177 h+1 i + vo  there poke
12 12 12 2 h+1 vo  there poke
13 13 13 h+1 h1 ! leave else
14 14 14 42 = if leave then then loop ;
15 15 14 0 \ capture - up_left, up_right  BTN 04-14-1988 \ capture - up_left, up_right  BTN 04-14-1988
1 2 : up_left vo 0 do  up_left  move one space up and one space left.
3 16 2 3 h-1 i - v-1 i - there peek vacant if
4 17 4 opoke 177 h-1 i - v-1 i - there poke
5 18 5 2 h-1 v-1 i - there poke
6 19 6 h-1 h1 ! v-1 vi ! leave else dup
7 20 7 42 = if drop leave then
8 21 8 186 = if leave then then loop ;
9 22 9 : up_right vo 0 do  up_right  move one space up and one space right.
10 23 9 10 h+1 i + v-1 i - there peek vacant if
11 24 10 opoke 177 h+1 i + v-1 i - there poke
12 25 11 12 2 h+1 v-1 i - there poke
13 26 13 h+1 h1 ! v-1 vi ! leave else dup
14 27 14 14 42 = if drop leave then
15 28 15 15 186 = if leave then then loop ;
15
"CAPTURE - DOWN_LEFT, DOWN_RIGHT"

0 \ CAPTURE - DOWN_LEFT, DOWN_RIGHT
1 : DOWN_LEFT 23 VO - 0 DO
2        H-1 I - V+1 I + THERE PEEK VACANT IF
3        OPOKE 177 H-1 I - V+1 I + THERE POKE
4                2 H-1 V+1 THERE POKE
5        H-1 H1 + V+1 V1 ! LEAVE ELSE DUP
6                42 = IF DROP LEAVE THEN
7                186 = IF LEAVE THEN THEN LOOP ;
8
9 : DOWN_RIGHT 23 VO - 0 DO
10       H+1 I + V+1 I + THERE PEEK VACANT IF
11        OPOKE 177 H+1 I + V+1 I + THERE POKE
12                2 H+1 V+1 THERE POKE
13        H+1 H1 ! V+1 V1 ! LEAVE ELSE DUP
14                42 = IF DROP LEAVE THEN
15                186 = IF LEAVE THEN THEN LOOP ;

16
"CAPTURE - FUNCTION"

0 \ CAPTURE - FUNCTION
1 : FUNCTION
2        : FUNCTION Check for the cursor key pressed and move accordingly.
3        DUP 72 = IF DROP UP^ EXIT THEN
4        DUP 80 = IF DROP DOWN EXIT THEN
5        DUP 77 = IF DROP RIGHT EXIT THEN
6        DUP 75 = IF DROP LEFT EXIT THEN
7        DUP 71 = IF DROP UP_LEFT EXIT THEN
8        DUP 73 = IF DROP UP_RIGHT EXIT THEN
9        DUP 79 = IF DROP DOWN_LEFT EXIT THEN
10       DUP 81 = IF DROP DOWN_RIGHT EXIT THEN
11       DUP 27 = IF DROP O END ! EXIT THEN
12       DROP 2 OPOKE ;
13
14
15
17
"CAPTURE - MOVE HERO and Misc"

0 \ CAPTURE - MOVE HERO and Misc
1 : MOVE HERO
2        : MOVE HERO Move the HERO. He can push obstacles.
3          KEY? IF EKEY DUP IXKEY ! ELSE IXKEY & THEN FUNCTION ;
4
5 : Y=0 4 RANDON 1 - 2/ Y ! ;
6
7 : X=0 4 RANDON 1 - 2/ X ! ;
8
9 : Y<0 00 IF -1 Y ! ELSE 1 Y ! THEN 100 RANDON DUP : Y<0
10       60 < IF DROP ELSE
11       75 > IF 0 Y ! ELSE Y & NEGATE Y ! THEN THEN ;
12
13 : X<0 00 IF -1 X ! ELSE 1 X ! THEN 100 RANDON DUP : X<0
14       70 < IF DROP ELSE
15       80 > IF 0 X ! ELSE X & NEGATE X ! THEN THEN ;
1. \CAPTURE - \MOVEB, \MOVE_BEAST, \DELAY & \SET_LEVEL BTN 04-14-1988 \CAPTURE - \MOVEB, \MOVE_BEAST, \DELAY & \SET_LEVEL BTN 04-14-1988

2. \MOVEB HB \@ HO - DUP 0: IF DROP X=0 ELSE X<0 THEN : \MOVEB Move the beasts to an empty space.

3. VB \@ VO - DUP 0: IF DROP Y=0 ELSE Y<0 THEN HB \@ X \@ + VB \@ Y \@ + THERE PEEK

4. BL = IF BL HB \@ VB \@ THERE PEEK

5. 42 HB \@ X \@ + VB \@ Y \@ + THERE PEEK

6. HB \@ X \@ + HB !

7. VB \@ Y \@ + VB ! THEN ;

8. 10 : MOVE_BEAST -1 CNT ! BEGIN 1 CNT +! MoveB : MOVE_BEAST Move the beasts. They cannot move obstacles, but must move around them.

9. CNT \@ \#BEASTS \@ 1- = UNTIL ;

13 : DELAY 25 SPEED \@ \# \@ 0 DO LOOP ; \Speed of game : DELAY Speed of game loop.

14 : SET_LEVEL DIFF_CNT DUP 1- DUP ROT ! ; : SET_LEVEL Set game difficulty level.

0 \CAPTURE - \LOSE? and \WIN? BTN 04-14-1988 \CAPTURE - \LOSE? and \WIN? BTN 04-14-1988

1. \LOSE? -1 CNT ! BEGIN 1 CNT +! \LOSE? Check for loosing the game.

2. HB \@ I 1- + VB \@ J 1- + THERE PEEK

3. 2 = IF 0 END \@ AT 4 24 ." "

4. 4 24 AT ." YOU LOSE \@ 0 LOSE ! LEAVE ELSE 1 LOSE !

5. THEN LOOP LOOP

6. CNT \@ \#BEASTS \@ 1- = UNTIL ;

7. 9 : WIN? -1 CNT ! \WIN ! BEGIN 1 CNT +! \WIN? Check for winning the game.

8. HB \@ I 1- + VB \@ J 1- + THERE PEEK

9. 32 = IF LEAVE ELSE 1 WIN +! THEN LOOP LOOP

10. CNT \@ \#BEASTS \@ 1- = UNTIL WIN \@ \#BEASTS \@ 9 \= IF 4 24 AT ." YOU WIN \@ 0 END ! THEN ;

11. 12

12. 13

14. 15

0 \CAPTURE - \Y/N, \TELL BTN 04-15-1988 \CAPTURE - \Y/N, \TELL BTN 04-14-1988

1. \Y/N BEGIN KEY UPC DUP 78 = IF 1 ANOTHER ! DROP EXIT THEN : \Y/N Check for a YES or NO answer.

2. B9 = IF -1 ANOTHER ! THEN ANOTHER \@ 0 <> UNTIL ;

3. 8

4. 5 : \TELL \TELL Display the instructions and ask for

5. CLS TITLE INVERSE \Instructions, \number of beasts, difficulty level

6. KEYS INVERSE1 \high-light \and

7. CONTINUE 0 24 AT 32 SPACES \#BEASTS? \DIFFICULTY? \SPEED? \initialization

8. FIND-LENGTH FILL-ARRAY ; \routine

9. 10

11. 12

13. 14

15
0 \ CAPTURE - PLAY

1 2 : PLAY

2   1 \ END : O XKEY : O ANOTHER : CLS \ Draw playing

3   4 COLOR BORDERS OBSTACLES BEAST HERO \ obstacles, the beasts and the HERO,

4   5 CONTINUE : O 24 AT 32 SPACES TIME! \ then play the game.

5   6 LEVEL : DIFFCNT !

6   7 BEGIN DELAY MOVE HERO \ Speed of game

7   8 SET_LEVEL 0= IF MOVE BEAST LEVEL \ Difficulty

8   9 DIFFCNT ! THEN LOSE ! LOSE \ and move/lose

9  10 0= IF ELSE WIN? THEN 24 AT TIME \ win

10 11 END 0= UNTIL 20 24 AT ; \ End game

11 12

12 13

13 14

14 15

0 \ CAPTURE - GAME

1 2 : GAME BEGIN TELL PLAY 50000 0 DO LOOP \ Play the game of Capture.

2 3 0 24 AT .Y Play again? (Y/N) *

3 4 Y/N ANOTHER 0= UNTIL CLS

4 5 BYE ;

5 6

6 7

7 8

8 9

9 10

10 11

11 12

12 13

13 14

14 15

0 \ CAPTURE - Build System

1 2 ONLY FORTH ALSO DOS ALSO \ search DOS and Forth

2 3 \ dummy program name

3 4 : CAPTURE EMPTY-BUFFERS \ parse filename to fcb

4 5 * CAPTURE.BLK FCBI (.FCB)

5 6 FCBI ! FILES OPEN-FILE \ open the file to list

6 7 GAME :

7 8 \ power up search order

8 9 ONLY FORTH ALSO \ make demo run automatically

9 10 \ capture.is.boot

10 11 SAVE-SYSTEM CAPTURE.COM \ create capture.com file

11 12

12 13

13 14

14 15

This screen is used to build a .com file of the program. This eliminates the need to compile (LOAD) the application each time it is needed.
Years ago, when structured programming was still relatively new, there was an article about goto-less programming using the "came from" construct. Since it appeared in the April issue (Datamation, I think), it made amusing reading and that was it. It is somewhat ironic, then, to be implementing a "came from" in what may be the most structured of all languages, Forth.

"Most of the CFA's were zero...a system reboot every time."

I've been doing some neat stuff with table-driven applications. For one of these, the basic data structure is a list of addresses of other data structures, which in turn point to other lists of yet other addresses; there are four levels of structures and address lists in total.

The implementation uses something like user variables, with a separate base pointer for each of the four levels. The addresses in the list are the CFAs (code field addresses) of words to initialize the base pointer.

The problem was that, during testing, the program kept trying to EXECUTE variables for which the base pointer had not been properly initialized. Most of the executed CFAs were zero. That meant a system reboot every time.

Hence the following test version of EXECUTE to check that the CFA to be executed is at least reasonable (i.e., within the application). It uses EXEC? to check that the address is in the range from 'EMPTY' to 'HERE'. If not, ABORT (see Figure One).

The next step was to show the return stack and the data stack just prior to ABORT by using .TRACE-BACK and .STACK. The word .SR shows the return stack, and is a simple transliteration of .S which, in F83, shows the data stack (see Figure Two).

The return stack, of course, contains (Continued on page 32.)
I, like me, you work odd hours and would like to somehow keep track of how much time you spend at your computer each week, you may find this program useful. It is written in F83 version 2.1 for the IBM PC and compatibles.

This program records the length of the current session, the total time for the day, the total for the current week (or other period of your choice), and remembers the total for the previous period. It also records the date and time in ASCII format when you log in or out, which can then be printed so that you have a record of exactly when you logged in and out.

How to Use It

Two files are used: TIMELOG.COM, the compiled form of the program, and TIMELOG, the file containing the logging data. I made TIMELOG three blocks long, which holds almost 50 pairs of login and logout times.

To register a login time, you type TIMELOG in from the DOS command line. Likewise, TIMELOG OUT will record the time you log out. Actually, just the letter I or O after the TIMELOG command will do the trick.

When you log out, it subtracts the time you logged in to compute the session time, and adds this to the daily total and the total for the period. To start from scratch, you would type TIMELOG R. This simulates a logout followed by a login, moves the current period total to the previous period, and then clears the current period total. The session total is always cleared by a login, and the daily total is cleared by a login if the day differs from the previous day of log in.

My work week ends at two p.m. on Thursdays, and I have therefore included another command: TIMELOG N. When I issue this command, the new week is started at two p.m. sharp (it gets the two p.m. from memory, not the clock.) Other than that, it works the same as the R command and could be used instead of it. The R command, however, is useful if you have been away for some days, and weren’t there on Thursday to add the total for the week; you’d start the new week right then. Of course, it will work just as well on periods longer than a week, but you should increase the TIMELOG file size accordingly.

If, during a session, you wish to know how much time you’ve put in, you can do so by using the command TIMELOG Q. This will display the data of any of the above commands, but won’t make any permanent changes to the logging file.

"These routines store the date and time into a log file that can be printed."

How It Works

My intention in writing this program was first to provide a useful application and, second, to provide clean and elegant code that may be of interest to readers. Being a firm believer in keeping things simple, I adopted the approach of deferring complexities at the higher levels of the code. It also seemed logical to list the program in reverse order, so that in explaining the program, the higher-level concepts are presented first.

The top-level word on screen 17 consists of just three parts: a beginning, a middle, and an end. That’s about as simple as it can get. The code at the bottom of the screen makes it possible to execute the program automatically from the DOS command line.

Screen 16 is where the different options get selected. I considered creating a case command for the occasion but decided against it, because it would only be used once and the code is pretty easy to follow anyway.

INIT on screen 15, again consisting of three words, gets your choice from the DOS command line, reads the log file, and gets the current date and time. The other two definitions on this screen aren’t quite as tersely defined, but nonetheless are pretty straightforward. One word worth noting is START, an F83 word which opens the DOS “default” file (normally obtained from the command line, but not in this case).

The code for NOW, on screen 14, is almost COBOL-like in its definition, where you would say something like: MOVE DATE TO PRESENT-DATE MOVE TIME TO PRESENT-TIME In Forth, of course, we use reverse notation, so the MOVE goes at the end and gets renamed to PUT because we already have a MOVE word in Forth.

NOW hides quite a lot of data: DATE@ and TIME@ each put four bytes on the stack, which are stored into the log file by PUT, using offsets defined by PRESENT, DATE, and TIME. I find the idea of using words to define offsets interesting and similar to adjectives in human language, in that they modify the object.
Also on screen 14, NEW.WEEK sets the present time to 14:00:00:00 hours (since my new week starts and ends at two p.m.) and then does a RESET, which does a simulated logout and login, moves the current week to last week, and zeroes the new week.

LOG.IN, on screen 13, checks if today’s date differs from the previous login date and, if so, clears the daily total. It also resets the session time and saves the login date and time. On the same screen, LOG.OUT and INQUIRE are almost identical, except that LOG.OUT saves the changes that were made, and INQUIRE doesn’t.

STRETCH, on screen 11, inserts 32 bytes at the start of block zero of TIME.LOG and moves everything else up. The last 32 bytes of the last block are saved just before the insert, since that is where the binary totals are being kept, and SAVE.END on screen ten is the word that puts those values back. This permits us to use the log file like a stack, with the most recent entry first. It is also independent of the number of blocks in the file.

Screens eight and seven deal with string handling. The purpose of these routines is to store the ASCII date and time into the log file in a format that can be printed out later, to provide something like an audit trail. The routines are commented pretty well. The only unusual definition is of $+C. SP@ gives the address of the character on the stack. Next, a one is pushed onto the stack, thus defining a one-character string which can then be processed by PLACE+ to be appended to the already existing string in INBUF.

Screens six and five contain logic to add with carry, and to subtract with borrow. The time calculations could, of course, be done by converting everything to hundredths of a second, but I consider this a more interesting way of doing it.

The routines that extract the time and date are on screen four. The time and date are converted into four separate bytes each, for uniformity and to simplify processing. Screen three contains some primitive words and screen two contains the constants and buffers used.

Final Note
I have used this program during the last few months and find it useful. One feature that could be added is a re-totalling option.

Occasionally I forget to log in or out, which throws off the totals. The log file can be edited, but I didn’t get around to writing a re-totalling routine yet. Perhaps you can come up with a neat way to do that?

TIMELOG.BLK Screen 17
\ LOG.TIME
: LOG.TIME ( S -- ) \ Top level time keeping word.
  INIT
  CHOICE
  FINAL ; \ Open log file, get input char.
  \ Decide what to do and do it.
  \ Wrap up the actions.

\ LOG.TIME IS BOOT \ for automatic execution

TIMELOG.BLK Screen 16
\ CHOICE
: CHOICE ( S char -- ) \ Input char determines what next
  DUP ASCII I =
  IF DROP " IN: " LOG.IN \ On login check daily total.
  ELSE DUP ASCII 0 =
  IF DROP " OUT: " LOG.OUT \ On logout update total also.
  ELSE DUP ASCII N =
  IF DROP NEW.WEEK
  ELSE ASCII R =
  IF RESET
  ELSE INQUIRE \ "Choices: In, Out, New, Query, Reset"
  THEN THEN THEN THEN ;

TIMELOG.BLK Screen 15
\ INIT GET.FILE FINAL
: FINAL ( S -- ) \ Display data and exit.
  CR 0 BLOCK $LEN 2* TYPE
  .TOTALS FLUSH 0 0 BDOS ; \ Display last 2 entries.
  \ Write data and exit.

: GET.FILE ( S -- ) \ Open time keeping file.
  [ DOS ] DOS-FCB CLR-FCB
  " TIME LOG" DOS-FCB 1+ 
  SWAP CMOVE START ; \ Get time keeping file name.
  \ Put file name in fcb & read it.

: INIT ( S -- char ) \ Initialize and get input char.
  DOS.CHAR
  GET.FILE
  NOW ; \ Get dos command line character.
  \ Open the time keeping file.
  \ Get the date and time.
TIMELOG.BLK Screen 14
\ NOW NEW.WEEK RESET
 : RESET (S -- ) \ Use this if starting afresh.
   " OLD: " LOG.OUT \ Close out last week.
   " NEW: " LOG.IN \ Start a new week.
   WEEK TIME LASTWK TIME XFER \ This week becomes last week.
   ZERO WEEK TIME PUT ; \ New week’s total is zero.
 : NEW.WEEK (S -- ) \ Do only after totals displayed.
   14 0 0 0 PRESENT TIME PUT. \ My new week starts at 2 pm.
   RESET ; \ Go and fix the totals.
 : NOW (S -- ) \ Get and save the date and time.
   DATE@ PRESENT DATE PUT \ (Only used at entry time.)
   TIME@ PRESENT TIME PUT ;

TIMELOG.BLK Screen 13
\ LOG.IN LOG.OUT INQUIRE
 : INQUIRE (S -- ) \ Session length = now – in time.
   PRESENT TIME LOGIN TIME SUB SESSION TIME PUT
   SESSION TIME DAILY TIME ADD DAILY TIME PUT
   SESSION TIME WEEK TIME ADD WEEK TIME PUT ;
 : LOG.OUT (S adr cnt -- ) \ Log out and add up totals.
   INQUIRE SAVE.TIMES ; \ Get elapsed time and log it.
 : LOG.IN (S adr cnt -- ) \ Log in date & time = now.
   PRESENT DATE DAY LOGIN DATE DAY DIFF?
   IF NEW.DAY THEN \ New day if not last login day.
   ZERO SESSION TIME PUT \ Start new session.
   PRESENT DATE LOGIN DATE XFER \ Update the time log.
   PRESENT TIME LOGIN TIME XFER SAVE.TIMES ;

TIMELOG.BLK Screen 12
\ NEW.DAY ZERO DIFF? XFER SAVE.TIMES
 : SAVE.TIMES (S adr cnt -- ) \ Save the log and update file.
   STRETCH SAVE.END \ Make room in file, save totals.
   MAKE$ PUT$ ; \ Put ascii log time in file.
 : XFER (S oal oa2 ob1 ob2 -- ) \ Transfer time from a to b.
   >R >R GET R> R> PUT ;
 : DIFF? (S oal oa2 oa3 ob1 ob2 ob3 -- t|f ) \ Compare 2 units.
   UNIT@ >R UNIT@ R> = NOT ; \ True if different.
 : ZERO (S -- 0 0 0 0 ) 0 0 0 0 ; \ Four zeros.
 : NEW.DAY (S -- ) \ Start a new day.
   ZERO DAILY TIME PUT ;

(Continued from page 29.)

the return address(es) in the calling word(s) and how we got to the point in the program where it failed. Doing DUMPS for each address on that stack to look for the name of the calling word was not, however, the way to go.

.CAME-FROM takes care of that (Figure Two). We know that the address on the return stack points in the calling definition following the two bytes whose content points to the CFA of the word currently executing (nested). Of course, there are occasionally values on the return stack that are not return addresses (e.g., after >R, loop control, etc.). There are also words like ?BRANCH and LITERAL that adjust the return address to skip past some in-line parameter. But by and large, if the address on the stack and the address it points to are both in a reasonable range, the latter probably points to a CFA.

The word .CAME-FROM does a reasonableness test (?EXEC) on the two addresses it uses and, if they both look good, uses >NAME .ID to go from CFA to NFA (name field address) and display the name.

Note that .CAME-FROM outputs something like ... 2A64=INTERPRET. That does not mean that INTERPRET is at 2A64. It does mean that the return stack contains 2A64, which corresponds to INTERPRET being the word that called us (at 2A62 we found the address of the CFA of INTERPRET).

(Continued from page 34.)
March/April 1989

THE BEST OF GENIE

GARY SMITH - LITTLE ROCK, ARKANSAS

Among the many things to which Forth is suited, real-time control has to rank as its strongest suit. Included in this ever-expanding category is the control of robots. Since the subject of robotics is near and dear to many Forth enthusiasts, it would be fair to expect discussion of such on GEnie Forth RoundTable. In fact, Category 6, Topic 9 is devoted to robotics; and in this issue we will sample the knowledge that is there for the asking.

Topic 9
Mon Sep 07, 1987 ATFURMAN [Alan F.]
Sub: Autonomous mobile & hobby robotics. This topic also includes legged locomotion.

Category 6, Topic 9, Message 2
Mon Sep 07, 1987 ATFURMAN [Alan F.]

Radio-Electronics magazine has been publishing a series of articles on building a mobile robot programmed in Forth (actually a robot-control wordset called RCL for "Robot Control Language") since December 1986. A kit of mechanical parts is offered by mail order. The robot has a one-degree-of-freedom "arm" (gripper on a vertical positioning slide). It has two powered wheels and a control board with an Intel 80186. The board is one of several SBCs sold bundled with Forth, and is made by Vesta Technology, Wheatridge, Colorado (which just happens to be run by the author, Steven E. Sams). Thanks to George Shaw for alerting me to this one.

I have not been following the series, so I do not know how sophisticated the software has gotten. This does seem like a great opportunity to blow the field away with some AI extensions to the Forth system. In fact, one of the niftiest AI hacks in Forth was created precisely for an autonomous mobile robot (at Oak Ridge National Laboratory; see "The Internals of FORPS: a FORTh-based Production System" by Christopher Matheus, Journal of Forth Application and Research, Vol. IV, No. 1, pp. 7-27 (1986). The inference engine and rule compiler take about one page of source and originally ran on a Z80. Now imagine running this on a Forth engine.

"A good feedback servo motor controller is a real thing of beauty..."

Category 6, Topic 9, Message 3
Tue Sep 08, 1987 S.W.SQUIRES [scott]

Versions of FORPS are available in the file directory for a few different computers. Search for FORPS.

Category 6, Topic 9, Message 4
Tue Sep 08, 1987 S.W.SQUIRES [scott]

I may be involved with a mobile cart for a future project. Since the wheels may be rubber, there will be some slippage. Does anyone know of good single-axis measurement-sensing system? Objective: the cart will be portable and will be in different environments with a minimal amount of setup time. Distance traveled will be from a few feet to 50 feet. Position resolution must be 1/100 of an inch or better. Update at a speed of 60 to 100 times per second. The technology must be practical with little maintenance. Possible ideas we’ve started to examine:

- Ultrasonics. Disadvantage: resolution and distance limited.
- Magnetic field. Disadvantage: not practical, given amount of metal and other factors.
- Visible focus (similar to Autofocus cameras). Disadvantage: complexity and limited resolution.
- Grid on ground. Disadvantage: not practical in environment.
- Tape on ground. Bar-code-style markings would be printed on the tape. This looks the most promising so far.

Although this is for a single axis, I’d be curious to know about three-dimensional measurements given the same criteria. — Scott

Category 6, Topic 9, Message 5
Tue Sep 08, 1987 ATFURMAN [Alan F.]

David Jaffe of the Palo Alto (California) Veterans Administration Hospital rehabilitation research group (and soon to be on GEnie) is connected with people doing mobile robot research at Stanford University. In particular, Larry Leifer of the Mechanical Engineering faculty.

Category 6, Topic 9, Message 6
Tue Sep 08, 1987 ATFURMAN [Alan F.]

Scott: Is this robotic cart connected with ILM? What is it for?

Category 6, Topic 9, Message 7
Wed Sep 09, 1987 S.W.SQUIRES [scott]
This would be a live-action dolly system with the same requirements as a normal dolly but be repeatable. This is just in an idea stage, so it might not become a reality. —Scott

Category 6, Topic 9, Message 8
Thu Sep 10, 1987 ATFURMAN [Alan F.]

All right. You guys want 0.25 mm. resolution over a 1.5–15 meter range of movement. I guess you are already aware of the avalanche of papers in the robotics literature on two subjects: navigation for mobile robots (obviously) and 3D sensors (for mobile robots and also for workpiece inspection and robotic bin picking). SPIE, SME, and IEEE run conferences with voluminous proceedings annually that address these topics. Here are a few thoughts:

Putting a target on the dolly and determining its position by triangulation requires a resolution of four seconds of arc. Theodolites are made that resolve 0.01 seconds, so it is potentially feasible.

A company called Digital Optronics is gearing up to commercialize very-high-resolution laser range finders. Rather than using time of flight (which performs poorly with attainable time resolution), these gadgets apparently chirp the (long-pulse) beam and heterodyne it with the return. Distance variations translate into frequency variations in the beat note. Clever, what? Of course, using it would still entail gimbal mounts and angular tracking of the dolly as it moves.

Coded tape is a contender, given the assumption that dolly motion is planned in advance, rather than arbitrary. More in our next episode...

Category 6, Topic 9, Message 9
Thu Sep 10, 1987 ATFURMAN [Alan F.]

Coded tape for repeat path sensing, continued from previous posting. Consider, if you will, the pattern [in Figure One] printed on, say, mylar tape.

A CCD camera aboard the dolly looks directly downward at this tape, which is stuck onto the floor. A simple image-processing algorithm locates a line crossing in the pattern, and compares its position in the image to the position seen during the lead-through programming run. Ambiguities as to which crossmark it
is are resolved by taking a cut (in software) through one of the bar codes (which label the scale every 1/10 meter).

Using one of the 512 x 512 sensors available now, and with optics imaging at 0.25 mm. per pixel, the camera will cover a field of 12.8 x 12.8 cm. For more coverage, sub-pixel resolution can be used. One approach to the latter is to use a fancier target pattern, as in Figure Two, for example, in which at least two lines are not parallel to the grid axes of the sensor are guaranteed. The software can then fit line equations to the diagonal pixel patterns (which smooth out the spatial sampling errors) and calculate their intersection. The software will also have to be smart enough to deal with wrinkles and overlapping ends of tape strips.

Cheers, Alan

Category 6, Topic 9, Message 10
Fri Sep 11, 1987 S.W.SQUIRES [scott]

Thanks for all the feedback Alan. Sounds like you're quite involved with these areas.

I looked at some of the optical and laser techniques when I was at the SPIE show this year. Would rather keep it a bit simpler. Your thoughts on tape are good. We were being read with a separate simple photodector. A single 'channel' of data should suffice if done correctly.

—Scott

Category 6, Topic 9, Message 11
Sun Sep 20, 1987 ATTFURMAN [Alan F.]

Since getting on GENIE, I have learned of the existence of the Macbot autonomous mobile robot project: a loosely defined, public-domain, hacker-community effort that seems to be drifting toward adopting Forth as the main programming language. References: several files in the Forth Applications DL, and postings under Category 12 in the "Mac developers" RoundTable.

The Macbot group figures that spindles alone (usable designs in servo control, AI, etc.) would pay for the effort, but practical

(Verhoeff screens, cont.)

TIMELOG.BLK Screen 7

```
\ INVERT PLACE+

: PLACE+ ( S from cnt to -- ) \ Like PLACE, but adds string.
  2DUP COUNT TUCK + -ROT + ROT C! \ Update count.
  SWAP CMOVE ; \ Then move string.
\ Example: if STRING contains 'Hello ',
  then "Joe" STRING PLACE+ will result in
  'Hello Joe' at STRING (last byte = count.)

: INVERT ( S a b c d -- d c b a ) \ Invert 4 items on stack.
  SWAP 2SWAP Swap ;
```

TIMELOG.BLK Screen 6

```
\ TIME+ +C ADD

: +C ( S x y w -- 0 x+y | 1 x+y-w ) \ Add w with carry.
  -ROT + 2DUP > \ Add a to b, compare with w.
  IF NIP 0 \ If sum < w means no carry.
  ELSE SWAP - 1 \ Else subtract w, carry = 1.
  THEN SWAP ; \ Keep sum on top.

: TIME+ ( S hi mi si .si ho mo so .so -- h m s .s ) \ Add times.
  4 ROLL 100 +C >R + \ Save .si + s.
  3 ROLL 60 +C >R + \ Save .so + si.
  ROT 60 +C >R + \ Save mo + mi.
  + R> R> ; \ Sub in hrs min sec secs/100.

: ADD ( S oal oal oal oal -- h m s .s ) \ Add times by offsets.
  >R >R GET R> R> GET TIME+ ;
```

TIMELOG.BLK Screen 5

```
\ TIME- -B SUB

: -B ( S x y w -- 0 x-y | -1 x-y+w ) \ Subtract with borrow.
  -ROT - DUP 0< \ Subtract y from x.
  IF -1 \ If negative, borrow w.
  ELSE NIP 0 \ Else get rid of w.
  THEN SWAP ; \ Keep difference on top.

: TIME- ( S hi mi si .si ho mo so .so -- h m s .s ) \ Out - in.
  4 ROLL 100 -B >R + \ Save .so - .si.
  3 ROLL 60 -B >R + \ Save .si.
  ROT 60 -B >R + \ Save mo - mi.
  SWAP 24 -B NIP R> R> R> ; \ Diff in hrs min sec secs/100.

: SUB ( S oal oal oal oal -- h m s .s ) \ Subtract a - b.
  2SWAP >R GET R> R> GET TIME- ;
```

(Screens continued on page 36.)
applications like aids to the handicapped have been mentioned. The only fixed quantity seems to be the Macintosh as central controller. Actually, the best controller choice would be the 32-bit Forth virtual machine, whether implemented on a Mac, Atari ST, Amiga, or 386 PC bus system (the last two choices would make interfacing easier).

Macbot activity on GEnie stalled this summer; I do not know what is going on with the project itself (it appears to live mainly on Compuserve). The leader of the project, B.W. Lightsey, is on GEnie; B.W.LIGHTSEY is his address.

Category 6, Topic 9, Message 12
Sat Nov 14, 1987 H.SIMMONS

Lacking practical experience, this suggestion may not have merit, but it would seem that the use of a fifth wheel would provide sufficient accuracy to allow for only occasional calibration by moving the dolly to a known location. If the dolly is to have two-dimensional travel, a larger-than-handheld “mouse” with larger ball should do the job.

For calibration, perhaps laser diodes attached to the ceiling at strategic positions, which could fire in response to an ultrasonic signal from the dolly? Is there any possibility of placing ultrasonic targets or “calibration tape” on the ceiling to get regular position information? —Horace

Category 6, Topic 9, Message 13
Mon Nov 16, 1987 S.W.SQUIRES [scott]

Those interested in Robotics and AI may want to check the October issue of OMNI magazine. There is an article on the insect robots designed by Rodney Brooks at MIT’s Artificial Intelligence Laboratory. The potential applications and approaches are discussed. The article in-
cludes step-by-step instructions on how to modify a cheap toy car from Radio Shack and add electronics to simulate some of the responses of an insect ($50-75 total). All the logic is actually photocells, Op Amps and TTL logic, but a person should be able to replace that logic with a small Forth board that does that and much more. As I recall, *Scientific American* had a “Computer Recreations” article a couple of years ago describing similar photocell-controlled vehicles. —Scott

Category 6, Topic 9, Message 15
Sat Dec 03, 1988 R.SCHEMMEL1 [JEPEDO]

Gentlemen, allow me to be of assistance. Robotics is my main objective and I would be glad to participate in a group project. I can provide technical reference information in electronics to almost any degree of detail, as far as circuitry goes. However, as I am sure you are aware, the sheer magnitude of new technology is staggering and, although I have a nice assortment of very recent engineering books on robotics-related subjects, nevertheless it is relatively equivalent to having a few fish out of the ocean: compared to what you don't have, you have nothing, but it's still enough to feed you...

Also, my forte is prototyping circuitry and fabrication of an electronic nature, to wit, building circuits, equipment, etc. including motor control. Feedback servos are, of course, the only way to get real precision but are incredibly involved, both design-wise and in prototyping and testing. A good feedback servo motor controller is a real thing of beauty: it flies through the air with the greatest of ease and stops close enough to a dime standing on its edge to knock it over with the air pressure and still not touch it. The precision standard four years ago was 1/1000 of an inch, but these days they have stuff that can thread a needle (literally) with plenty of clearance—if you have enough money. Anyway, if you need something prototyped and have the design but need an engineering technician to build it, I might be able to help you. I am hoping I'll eventually learn enough about Forth to program the hardware I build, but at the moment I couldn't program my way out of a paper bag.

Personally, I'll take a Forth engine over anything else for a main CPU or control processor. I just wish I could afford one!

(Verhoeff screens, cont.)

TIMELOG.BLK Screen 0
\ Time keeping program TIMELOG.BLK

Copyright (c) 1988 by Peter Verhoeff
308 N. Louise Ave. #14
Glendale CA 91206

This program is freely available for private use. Commercial use of this program or any part thereof requires written permission from the author.

This program has been written in F83 Forth, version 2.1.0.

(Continued from page 4.)

Other Forth-specific BBS's
- Laboratory Microsystems, Inc.
  213-306-3530
SysOp: Ron Braithwaite

This list was accurate as of March 1989. 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: 501-228-0271
Telex: 6501165247 (store and forward)
GEnie (co-SysOp, Forth RoundTable): GARY-S

BIX (Bytenet): GARYS
Delphi: GARY_S
MCIMAIL: 116-5247
CompuServe: 71066,707
Wetware Diver. (Fairwitness, Forth Conference): gars
Usenet domain: gars@well.UUCP or gars@wet.UUCP
Internet: well!gars@lll-winken.arpa
WELL: gars

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

Volume X, Number 6 37 Forth Dimensions
Scott, for your information the RE-ROBOT uses an 80188, not 80186 as you mentioned. Also, for those interested, the RR-BBS phone number is 516-293-2283. It was busy every time I called, but they have a section for RE-ROBOTEERS, so to speak.

Also, on the subject of rubber wheel slippage I would suggest you forget that word “slippage” because all you really care about is not how much you’ve slipped, but where you are when all the slipping is over. That is, concentrate on getting your bearings using something like the Polaroid Infrared range sensor for course measurements and a doppler-type ultrasonic sensor for a complementary input. You’ll need something photoactive for fine distance measurements and I would recommend a high-powered infrared LED mounted on a one-axis mount or a disk such that it can change the angle at which it shines on the path or the wall. Have a strip of infrared optical transistors running along the base strip all the way around so the transmitter scans back and forth, sending a pulsed beam at a preset frequency which the transistors receive. Multiplex the transistor detectors for a signal and test the signal found for the correct frequency to eliminate all light not sent by the transmitter.

Admittedly this is a crude method, but the key to making it work is using linear transistors and sensing amplitude of return signal rather than switching type in an on/off setup. By varying the current to your scanning transmitter LED and sensing linearly varying light pulses returning, you have the ability to make either “short range” or “long range” sensor scans. By using the amplitude of the return light signal to vary the frequency of a voltage-to-frequency converter IC (there are many available), you can measure the frequency of the signal you have generated and use that when you wish to know the distance.

The cart sensor navigation system is calibrated by driving it in a learning mode through an obstacle course and recording all sensor readings. The measurements are used to establish equivalent parameters such that a data value \( X \) equals ten feet, or six feet, or one foot etc. The readings, of course, are never the same for different areas so, given a large memory capacity for read-only data (CD ROM would be ideal, but a large hard disk will do), you can record all the readings for points along the path and store them once as data — this is a map which has no value to anyone except the robot that generated it, to which these readings represent real places it has been to once. Therefore, it can do a string search or approximation comparison of real-time readings with its stored map data and determine that it is about three feet from the drinking fountain in front of the elevator door on the second floor near the east wing at about 4:00 p.m., when the sun coming...
through the window is at its lowest. As you can see, place is only part of the problem; the time must be recorded when the map is generated. That is, the data header for the sensor readings must always contain a date-stamp or the readings will lose much of their value. As a matter of general practice, all sensor readings of any nature should contain a date stamp, as this information will be invaluable later.

"Jepedo is the name and robots are my game."

---

**Figure One.** Ruled tape with bar code labels at ten-centimeter increments.

---

**Figure Two.** Targets used to achieve sub-pixel resolution require calculation.

---

**THE SUPER8 DEVELOPMENT LAB**

An unparalleled development environment for a whole spectrum of applications

Complete documentation for Super8 and FORTH use

$295

single quantities

**SUPER8 DEVELOPMENT BOARD,** with Zilog's powerful 20MHz Super8 single-chip microcomputer — with

- monitor ROM for conventional assembler development
- FORTH ROM set for interactive FORTH development with full implementation of F83 FORTH
- Prototyping area

**PC Terminal Emulation and Disk Server software**

Super8 assembler/utilities software

Super8 monitor/instructions/examples software

**Inner Access Corporation** Box 888 Belmont, CA 94002 · (415) 591-8295 · Telex 494-3275 INNACC
THE GREENING OF FORTH

JACK WOEHR - 'JAX' ON GEnie

Many proverbs of our civilization indicate that he who wishes to lead must discover in which direction the people are heading and then place himself in front of them.

The Forth Interest Group has been a leader of the steady and inspiring progress of the Forth programming environment for well over a decade. Recently, however, the question arises, "Is anyone still following?"

Forth is more popular than ever. The surest gauge of the health and well-being of Forth is to be had in the number of annual announcements of its impending death. Last year was perhaps a banner year for Forth, its demise being proclaimed in forums where its name had not previously been heard, such as John Dvorak’s writings.

Forth in 1988 acquired a powerful and influential patron in Harris Semiconductor. No less than Dr. C.H. Ting has testified that, "I used to think I knew all the preachers of Forth," but after attending a Harris RTX2000 seminar, he owned that he had heard the gospel preached eloquently by well-informed individuals previously unknown to the small and cozy Forth community.

Forth continues to sweep the field of microprocessor-based embedded systems. FORTH, Inc. announced in 1988 several innovative extensions of their line of embedded-system-targeted polyFORTHs.

As an embedded-system programmer myself, I have daily phone or BBS contact with inventors, scientists, designers, and engineers eager to acquire a working knowledge of Forth. They all have come to the conclusion that, like it or not, they must learn Forth to get the job done in reasonable time for a reasonable investment of cash.

A class I teach in Golden, Colorado continues to gain new attendees, individuals who have discovered Forth on their own and seek tutoring in the basics.

An admittedly unscientific and subjective assessment of job opportunities for Forth programmers leads me to believe that there have never been more positions for qualified individuals in the history of Forth, nor have so many of these positions ever gone begging for so long. Furthermore, Forth is reaching higher in the corporate world; for example, IBM here in Colorado is committing major programmer effort to the Forth-coded IBM-CAD project.

Yet those of us who enjoy the FIG fellowship must find it subject for concern that this "Greening of Forth" does not seem to be entirely reflected in the Forth Interest Group. Membership has certainly not increased in proportion to the Forth boom; rather, we find old members too busy to participate any longer, often less enchanted with Forth Dimensions, occasionally disgruntled at actions, or lack of same, from the FIG leadership.

To a certain extent, this falling away from FIG is limited to North America. I am informed by telecom friends in Europe and Australia that new Forth Interest Group chapters are constantly being formed, and that the chapters that exist are lively and well attended. Recently, we certified the first chapter in Finland, with a second possibly to follow; new applications have reached Kent and Jan in the business office from as far away as Bulgaria!

If FIG indeed is currently in decline, what could be the reasons?

First of all, the institution of the Saturday computer club is in decline in North America. Computer savvy goes crying in the streets nowadays; we are surrounded by all the digital wizardry we could desire and more; the beginner does not need as much hand-holding from monthly meetings as before, since the sophistication of even the most casual computer user is far greater than that of ten years ago.

Secondly, while FIG has made one bold stride in the direction of the brave new world by opening the Forth Interest Group RoundTable on GEnie, other services are needed by the modern Forth programmer that FIG has not yet been able or seen fit to provide. It may yet be that other organizations will step in to fill the vacuum, such as the Association for Computation Machinery’s SIG-Forth, which sees itself as a natural forum for the professional Forth programmer.

Also, FIG has been drifting slightly under the influence of a quite natural process of new faces coming into the organization to replace experienced hands, your correspondent finding himself among the former culpable grouping.

Many suggestions have been offered, many plans have been laid for the continuation of the work of the Forth Interest Group. Those interested in summaries of these suggestions, or in making suggestions of their own, will find the most ready audience waiting in the various telecom institutions growing up with Forth—see the "Reference Section" elsewhere in this issue. Furthermore, any chapter can subscribe to our monthly Chapters Newsletter by contacting me on any of those services or at my UUCP address of jax@well.UUCP, well!jax@ill-

Forth Dimensions 40 Volume X, Number 6
winken.arpa, or alternatively by shouting for my attention on USENET's comp.lang.forth discussion group.

Your Chapter Coordinator's personal offering to the FIG suggestion box is tried, trite, and true: the future is in the hands of youth. FIG has yet to try a major outreach to high schools in every city in the world where there exists a FIG chapter. IMHO (as we say in telecom) such a framework for the propagandaization of the youthful entrant to the digital universe is long overdue.

I feel very optimistic about FIG. The Forth Interest Group, in particular the Silicon Valley FIG Chapter, was instrumental in my entry into the world of Forth. Furthermore, since May 1987, I have held three consecutive full-time Forth programming positions, two obtained through FIG meetings, the latest and current position here in Colorado found in the "Programmer Wanted" ads on the GEnie FIG RT.

I see no reason why the Forth Interest Group cannot continue to offer a friendly and helpful gateway to Forth for the beginner while serving the changing and varied needs of the professional. Innovation and dedication will be required from the members of the world's oldest fraternal association of Forth programmers, which will no doubt be available in abundance, since are not innovation and dedication the very hallmarks of the Forth programmer?

1988 PROGRAMMERS CONTEST

Held at last year's FIG-sponsored "Real-Time Programming Convention," this event captured the spirit of the Forth programmer confronted with an unusual problem. The object of the contest was a closely held secret until the event began.

Winners of the contest were Phil Burk and Mike Haas from Delta Research.
The FIG Chapters listed below are currently registered as active with regular meetings. If your chapter listing is missing or incorrect, please contact Kent Safford at the FIG office's Chapter Desk. This listing will be updated in each issue of 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
      - Horace Simmons
      - (907) 486-5049
  - **ARIZONA**
    - Phoenix Chapter
      - 4th Thurs., 7:30 p.m.
      - AZ State University
      - Memorial Union, 2nd floor
      - Dennis L. Wilson
      - (602) 956-7578
  - **ARKANSAS**
    - Central Arkansas Chapter
      - Little Rock
      - 2nd Sat., 2 p.m. & 4th Wed., 7 p.m.
      - Jungkind Photo, 12th & Main
      - Gary Smith (501) 227-7817
  - **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., 10 a.m.
      - Fort, Al 12 Noon Tutorial, 1 p.m.
      - Fort South Berkeley Public Library
      - George Shaw (415) 276-5953
    - 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
      - Tom Ghormley (916) 444-7775
    - San Diego Chapter
      - Thursdays, 12 Noon
      - Guy Kelly (619) 454-1307
    - Silicon Valley Chapter
      - 4th Sat., 10 a.m.
      - H-P Cupertino
      - Bob Barr (408) 435-1616
    - Stockton Chapter
      - Doug Dillon (209) 931-2448
  - **COLORADO**
    - Denver Chapter
      - 1st Mon., 7 p.m.
      - Clifford King (303) 693-3413
  - **CONNECTICUT**
    - Central Connecticut Chapter
      - Charles Krajewski
      - (203) 344-9996
  - **FLORIDA**
    - Orlando Chapter
      - Every other Wed., 8 p.m.
      - Herman B. Gibson
      - (305) 855-4790

- **SOUTHEAST FLORIDA CHAP**
  - **ORLANDO**
    - Every other Wed., 8 p.m.
    - Herman B. Gibson
    - (305) 855-4790
  - **TAMPA**
    - 1st Wed., 7:30 p.m.
    - Terry McNay (813) 725-1245
  - **GEORGIA**
    - Atlanta Chapter
      - 3rd Tues., 6:30 p.m.
      - Western Sizzlen, Doraville
      - Nick Hennenfent (404) 393-3010
  - **ILLINOIS**
    - Cache Forth Chapter
      - Oak Park
      - Clyde W. Phillips, Jr.
      - (312) 386-3147
    - Central Illinois Chapter
      - Champaign
      - Robert Illyes (217) 359-6039
  - **INDIANA**
    - Fort Wayne Chapter
      - 2nd Tues., 7 p.m.
      - I/P Univ. Campus, B71 Neff Hll
      - Blair MacDermid (219) 749-2042
  - **IOFA**
    - Central Iowa FIG Chapter
      - 1st Tues., 6:30 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) 472-7077
  - **MARYLAND**
    - MDFIG
      - Michael Nemeth
      - (301) 262-8140
  - **MASSACHUSETTS**
    - Boston Chapter
      - 3rd Wed., 7 p.m.
      - Honeywell
      - 300 Concord, Billerica
      - Gary Chanson (617) 527-7206
  - **MICHIGAN**
    - Detroit/Ann Arbor Area
      - 4th Thurs.
      - Tom Chrapkiewicz
      - (313) 322-7862
  - **MINNESOTA**
    - MNFIG Chapter
      - Minneapolis
      - Even Month, 1st Mon., 7:30 p.m.
      - Odd Month, 1st Sat., 9:30 a.m.
      - Fred Olson (612) 588-9532
      - NC Forth BBS (612) 483-6711
  - **MISSOURI**
    - Kansas City Chapter
      - 4th Thurs., 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 Lordi
      - (201) 338-9363
Volume X, Number 6

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

- NEW YORK
  FIG, New York
  2nd Wed., 7:45 p.m.
  Manhattan
  Ron Martinez (212) 866-1157

- ROCHESTER
  Odd month, 4th Sat., 1 p.m.
  Monroe Comm. College
  Gary Bergstrom (216) 247-2492

- PENNSYLVANIA
  Villanova Univ. FIG Chapter
  Bryan Stueben
  321-C Willowbrook Drive
  Jeffersonville, PA 19403
  (215) 265-3832

- TENNESSEE
  East Tennessee Chapter
  3rd Tues., 7 p.m.
  800 Oak Ridge Turnpike
  Richard Secrist
  (615) 689-8161

- TEXAS
  Austin Chapter
  4th Thurs., 7:30 p.m.
  Texas Instruments
  13500 N. Central Expwy.
  Semiconductor Cafeteria
  Conference Room A
  Cliff Penn (214) 995-2361

- HOUSTON
  3rd Mon., 7:45 p.m.
  Intro Class 6:30 p.m.
  Univ. at St. Thomas
  Russell Harris (713) 461-1618

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

- RICHMOND
  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
  03/29/2600
  BBS: 61 3 299 1787

- SYDNEY
  2nd Fri., 7 p.m.
  John Goodsell Bldg., RM LG19
  Univ. of New South Wales
  Peter Trequagne
  10 Binda Rd., Yowie Bay
  2228
  02/524-7490

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

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

- CANADA
  Northern Alberta Chapter
  4th Sat., 10 a.m.–noon
  N. Alta. Inst. of Tech.
  Tony Van Muyden
  (403) 486-6666 (days)
  (403) 962-2203 (eves.)

- CANADA
  Southern Ontario Chapter
  Quarterly, 1st Sat., Mar., Jun.,
  Sep., Dec., 2 p.m.
  Genl. Sci. Bldg., RM 212
  McMaster University
  Dr. N. Solntseff
  (416) 525-9100 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
  Pn FIG
  Jaume Kotiranta
  Arkkitehdinkatu 38 c 39
  33720 Tampere
  +358-31-184246

- HOLLAND
  Holland Chapter
  Vic Van de Zande
  Finmark 7
  3831 JE Leusden

- ITALY
  FIG Italia
  Marco Tausel
  Via Gerolamo Forni 48
  20161 Milano
  02/435249

- JAPAN
  Japan Chapter
  Toshi Inoue
  Dept. of Mineral Dev. Eng.
  University of Tokyo
  7-1-1 Hongo, Bunkyo 113
  812-2111 x7073

- N. AMERICA
  Columbus FIG Chapter
  Terry Webb
  (614) 877-7241

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

- OREGON
  Williamette Valley Chapter
  4th Tues., 7 p.m.
  Linn-Benton Comm. College
  Pann McCuaig (503) 752-5113

- PENNSYLVANIA
  Villanova Univ. FIG Chapter
  Bryan Stueben
  321-C Willowbrook Drive
  Jeffersonville, PA 19403
  (215) 265-3832

- TENNESSEE
  East Tennessee Chapter
  3rd Tues., 7 p.m.
  800 Oak Ridge Turnpike
  Richard Secrist
  (615) 689-8161

- TEXAS
  Austin Chapter
  Matt Lawrence
  PO Box 180409
  Austin, TX 78718
New from the Forth Interest Group

Proceedings of the First Australian Forth Symposium
Organized by Forth users from industrial and academic organizations. Held May 19–20, 1988 at the University of Technology Sydney School of Physical Sciences. 154 pages.

$24.00

Tenth Annual FORML Conference
These proceedings contain papers from the FORML Conference held November 25–28, 1988 at the Asilomar Conference Center, Pacific Grove, California. 156 pages.

$24.00

Special introductory offer available—buy both books together—total price $40.00.

Mark Order Form: FORML Special.

FIG members entitled to additional membership discount—see order form.

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