$10 Volume XIX, Number 4 November 1997 December

FORTH DIMENSIONS

MicroFont: 4x5

Misty Beach Forth: An Implementation in Java

Limits to Growth

Forth in Control: Analog/Digital Converters

OFFICE NEWS

What a year of transition! So much has transpired this past year, I thought I'd take a few extra lines and share it with you.

January—the FIG business and sales office moved from its established home in Oakland, California to Carmel, California. Frank Hall retired as the Office Administrator after many years of fine service to the FIG community.

February—200 boxes of back inventory was moved to Monterey. We managed to do this over a weekend. John and Frank Hall, Skip Carter, and I moved each and every box. If you need to fill in your library with back years of *Forth Dimensions*, you can do it through FIG. We even have a good supply of FORML Conference proceedings. As we downsize, this inventory will not always be available; if you want it, now is a good time to get it.

March—the new FIG office started filling orders for books and software, in addition to processing renewal memberships and new memberships. The majority of our members come up for renewal in March, so you can imagine the deluge of paperwork that came our way. Thankfully, at that time I still had Julie Stone working in the office. Julie started with me in January to help get things organized. Her presence is greatly missed.

In March, the new FIG office successfully bulk mailed our first issue of *Forth Dimensions*. We held our breath, hoping we had done everything right that first time and that every FIG member would get their issue as planned. In fact, Frank was generous enough to come back for a day and help with the bulk mailing procedures to see that we were on the right track.

April—more processing! More sorting! More archiving of files! April was the month of trying to get our feet firmly planted on the ground and to achieve a smoothly running office. We went on-line with e-mail for the first time in our new location! Now, not only were we processing orders and renewals from mail-in responses, but the Internet was producing 10–20 e-mails a day! Believe me, for a first-time user of e-mail, that alone can be intimidating.

May—I don't remember anything spectacular happening in May, I think we just worked hard to get things caught up. And, actually, I'm rather grateful for that.

June—I decided to attend the Rochester Forth Conference for the first time. It was great to meet more FIG members and, as I've stated before, Larry and Brenda Forsley are wonderful hosts. The Forth community is truly fortunate to have two great conferences here in the U.S. to attend.

July—Back at the office, while I was still on the East coast, Julie with the enlisted help of Brandon Yeager (many of you know his name from e-mails; his time has been volunteered as System Administrator of the FIG web site, in addition to helping those of you who have trouble from time to time getting on the "Member's Only" section) successfully compiled and bulk mailed another issue of *Forth Dimensions*.

August—New levels of membership have been established! Businesses working with Forth and profiting from it are willing to step up and support FIG at a higher level. The advantage for the business is increased visibility on the web site and in *Forth Dimensions*. The advantage for FIG is a much-needed increase in funds. Thank you to all those first Corporate Members, you are the leading edge of a new level of support for the Forth Interest Group.

You'll notice in the last issue of *Forth Dimensions*, as well as this issue, the listing of Benefactor Membership. John Hall wanted to make known his willingness to support FIG financially at the same level as the Corporate Members, but wanted to do it as an individual: thus, the Benefactor level has been established. Once again, a way to donate to FIG and get recognized for that donation. Please, if you're able to do this, contact me; we can convert an existing standard membership to Benefactor status.

September— An exciting time! For the first time since January, we have increased our membership. Membership in the Forth Interest Group had been on a decline. With the addition of Corporate Memberships and the new Benefactor Membership level, FIG has new levels of financial support. With a smaller membership and the continued support of higher-level memberships, we can continue to bring you the quality product of *Forth Dimensions*, hopefully for years to come.

October—time to prepare for FORML, a first for this office. Each year, Bob Reiling undertakes the mammoth job of making the preparations for this conference held at the Asilomar Conference Center in Pacific Grove, California. Have you attended? It's a fabulous place to meet the innovators of Forth. You can meet and talk with Chuck Moore, Wil Baden, Elizabeth Rather, and Skip Carter, to name just a few.

November—it's FORML time. Not only did we get another issue of *FD* out the door just a week before the conference, everything managed to go off without a hitch. Keep an eye on the web site (www.forth.org) for new developments to be announced for next year's FORML conference.

December—time to reflect on this past year. Soon we'll be adding a Volunteer Project List to the web site. The Forth Interest Group has more projects than we at the office can possibly do alone. We need *your* help to increase our membership, be it in time or money; now is the time to volunteer one or the other to FIG.

As always, we at the FIG office enjoy hearing from you, by phone, by fax or by e-mail. This next year will prove pivotal to the continuation of the Forth Interest Group—please be part of it!

—Trace Carter
office@forth.org

CONTENTS

EDITORIAL

Largeness

Every issue of this magazine reminds me of the generous nature of its many contributors. I wish to voice a collective thanks to each of our authors, for whom the recognition of their peers and the knowledge that they have made a tangible contribution are their primary compensation. And each reader supports this effort by keeping an active membership, for which we remain grateful while striving to deliver at least an equal value in content each year.

Grass-roots participation has always powered this organization and, arguably, the Forth language itself. The FORML Conference, the FIG Chapters, euroForth, the Rochester Forth Conference, and the various electronic venues of Forth information and interchange all thrive in the fertile fields of volunteer effort and enthusiastic participation. In my opinion, the participation itself is a reward, beyond the knowledge shared and gained, the personal and professional contacts made, and the enhancing effect on Forth careers and avocations.

I don't know who wrote the quotation shown on this page; it arrived via e-mail from an unattributed source and has served as a salutary reminder.

"Volunteers are individuals ... who reach beyond their paid employment ... to contribute time and service ... in the belief that their activity is beneficial to others ... as well as satisfying to themselves."

It was great to be at FORML once again, this year. The report which begins on the facing page conveys something of the conference's tone as well as content; those who are accustomed to attending find it satisfying in many ways that I hope others of you will be able to discover starting next year. Keep the dates open—you will be hearing more from us in coming months, and from the way it is already taking shape, I suspect you will want to seriously consider attending even if you have not before.

For this issue, Skip Carter takes a break from his Forthware column (so rest your math muscles) to share the paper he presented at FORML. Those who have long wondered at the relatively slight use of Forth for large software projects will find what is surely a comprehensive rationale and at least part of the remedy.

We are pleased to welcome Ken Merk back to our pages, with another in his occasional series about controlling hardware with Forth. We know Forth is used in many embedded and control situations—it has been claimed that, if one counts by the number of actual devices manufactured instead of by computers with keyboards and disk drives, Forth might be *the* most-used software in the world. But it isn't always easy to learn how to achieve hardware control if one doesn't have a certain amount of experience. Ken's articles are a help to the hardware-challenged, and I am pleased to hint (but only a hint, for now) at other work in development that will provide additional, expert help.

Loosely related to Richard Astle's remarks about implementing Forth, we find discussion about a developing Java implementation. To date, most discussion in the Forth community about Java has had the characteristic of drawing easy parallels; this author has more experience with Java than with Forth, and brings a different perspective as well as concrete approaches, goals, and comparisons. It isn't as easy as those simple comparisons would have it, but imagine downloading a Forth applet to any Java-enabled browser—not only to provide new functionality in the browser environment, but to offer practical demonstrations of what we're talking about to a large audience that is, largely, willing to try "new" things. We think this is an interesting project, with plenty of thorny performance issues to (try to) resolve; let us know what *you* think.

Marlin Ouverson
editor@forth.org

Forth Dimensions
Volume XIX, Number 4
November 1997 December

Published by the
Forth Interest Group

Editor
Marlin Ouverson

Circulation/Order Desk
Trace Carter

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

Subscription to *Forth Dimensions* is included with membership in the Forth Interest Group at $45 per year ($53 Canada/Mexico, $60 overseas air). For membership, change of address, and to submit items for publication, the address is:

Forth Interest Group
100 Dolores Street, suite 183
Carmel, California 93923
Administrative offices:
408-37-FORTH Fax: 408-373-2845

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

FORTH DIMENSIONS (ISSN 0884-0822) is published bimonthly for $45/53/60 per year by Forth Interest Group at 1340 Munras Avenue, Suite 314, Monterey CA 93940. Periodicals postage rates paid at Monterey CA and at additional mailing offices.

POSTMASTER: Send address changes to FORTH DIMENSIONS, 100 Dolores Street, Suite 183, Carmel CA 93923-8665.

FORML '97 REPORT

I'm writing this in the time slip between the 19th FORML conference and the weekend it would have been by last year's schedule. I would have been driving north now, on 5 towards Los Angeles, full of Thanksgiving dinner, on my way to Monterey, Pacific Grove, and Asilomar by the sea.

The switch from Thanksgiving weekend to the weekend before had its effects, besides giving repeating conference attendees a sense of dislocation: lines for lunch and dinner were shorter, and parking was easy in Carmel. The change may have kept some from coming (costing a vacation day) but it also meant the presence of some who have other things to do on holidays. Forth, Inc.'s Elizabeth Rather, the "second Forth programmer," attended for the first time, though some of us were surprised when she said so, having seen at least her ghost there previously.

Asilomar and FIG are both under new management this year, and though FIG is none the worse for it, the same cannot quite be said of Asilomar, though noticeable ill effects were small: new rules mean box lunches have to be ordered two days in advance, and bartenders must be hired to monitor corkage fees at our traditional evening wine-and-cheese parties. FORML has a life of its own, however, and a continuity provided by Bob Reiling's directorship which allows that life. Bob opened the conference with reminiscences, and showed photos from former years, including one of a bushy Marlin Ouverson with, I'm convinced, a clean-shaven me in the background.

FORML kept its usual schedule: papers presented in two afternoon and one evening session on Friday and two morning sessions on Saturday, working groups Saturday afternoon, impromptu talks Friday evening, closing remarks, awards (bottles of wine for categories invented at the last minute) and a panel discussion Sunday morning, group meals all days, and wine-and-cheese parties (accompanied by various hard- and software demonstrations) both evenings. In the gaps, we walked the boardwalk and the beach, watched waves and deer and snakes, and generally enjoyed ourselves.

Sessions (and parties) were held this year in Heather, a meeting room in the basement of one of the newer buildings in the center of the Asilomar grounds, where we last met about fourteen years ago, my first FORML. There were passions then, talk about Neon, which left Forth to become Actor, mention of that great almost successful Forth project Valdocs, Leo Brodie, not enough chairs, a sense of excitement. The Forth community has matured, or aged, sobered up a little (even the late night sessions are tamer), and shrunk (if conference attendance is an indication). There were two dozen of us this year, mostly familiar faces, mostly Californians.

Sessions were chaired, efficiently and more gently than would have been possible in the past, by Guy Kelly. There were more pre-submitted papers than last year (there could hardly have been fewer), and at the first session we started going right through them. One of the most interesting papers was the first, Wil Baden's discussion of an implementation of an encryption scheme he calls "Arcipher," also discussed in his column in the issue of *Forth Dimensions* we received with our registration packets. He was given an award (he gets one most years) for "the most immediately applicable nugget of code." Wil's other papers (he always brings several) were on Formula Translation, which has something to do with something called "Fortran," and "Semi-literate Programming," which describes his method (an excursion into territory previously colonized by Glen Haydon) of mixing code and comments in a single text file so it can be compiled and printed several different ways for different purposes. This last paper provoked Dr. Ting to remark that Wil has too many ideas for the rest of us conveniently to absorb. Wil's eventual response to that remark was that if he sees a fat rabbit crossing the road he's going to chase it.

If you get the idea that the conference was largely a group of old friends sitting around talking, or standing up in front of each other and talking, that would describe about half of it.

Another award came out of that first afternoon session for, if my notes are correct, "Just a ROM to run the dog." It was Dr. Ting's turn to bring a robot this year, and he brought "Gogo: a Mindless Robot." Unlike Skip Carter's six-legged walker (featured in *Dr. Dobb's* after last year's conference), Dr. Ting's rolled, controlled by bumper sensors and a state-machine, but it was strong, powered by a car battery and wiper motors and, though it threw a wheel, it didn't break a leg.

Between the first two sessions, Friday afternoon, we checked into our rooms and, though mine was on the second floor, with no deck and an interior hallway, I saw a deer wander through the trees below my window.

The second afternoon session had three papers: one of Wil's, a typically apt Forth philosophy talk by Glen Haydon, and another prize winner (for "outstanding contribution to the future of Forth"): Skip Carter's "Limits to Growth." Skip's main point was that a language's available toolset, more than its syntactic or semantic power, determines what it is likely to be used for. So, according to Skip, assembly language is used for small embedded systems projects, Forth for mid-sized projects, and C/C++ for large projects. Elizabeth Rather questioned whether this were a "political rather than a technical" issue but continued the theme in the evening session when she announced Forth, Inc.'s forthcoming "SwiftForth," a Windows-hosted Forth system and environment, with subroutine-threaded code, in-line substitution, dialogs, and menus. Not only does this Forth promise, according to benchmarks, to compile faster code, in most cases, than polyForth, MPE, and Win32Forth, it will also provide at least some of the development tools taken for granted by C++, Delphi, and even Visual Basic programmers. Although it won no wine award, this announcement seemed to provoke the most interest. In the final conference session many of us expressed a desire to investigate (i.e., "play with") SwiftForth, at least if the price is right, and only one expressed a refusal to "contribute to the dark side."

The Forth world has a number of icons, names, figures everybody knows or knows about: Laxen and Perry, who wrote F83; Tom Zimmer, the principle architect of F-PC and Win32Forth; Bill Ragsdale, from the early days; Chuck Moore,

Richard Astle • Del Dios, California
rastle@bigfoot.com

Richard Astle has a bachelors degree in math from Stanford University, a master's in creative writing from San Francisco State, and a Ph.D. in English literature from the University of California (San Diego).

of course, who gave us all of this like Little Richard gave us rock-and-roll; and Elizabeth Rather's Forth, Inc., the main bastion of professionalism in a field often characterized as amateurish and maverick. Everybody knows that Forth's greatest strength, its ease of implementation, is also its greatest weakness ("if you've seen one Forth, you've seen one Forth"), leading to mutual incompatibility and an inability to support compiler vendors. However much C/C++ programmers might complain about Microsoft's and Borland's compilers, the fact that they have to wait for the next release for bug fixes keeps them united. Perhaps, with Windows and other complications, the computing world, for better or worse, has finally gotten to the point that a Forth vendor can make money selling systems to the rest of us, in which case we'll be glad Forth, Inc. has survived this long. I know I'll buy SwiftForth, if I can afford it.

Other papers in the Friday evening session included Dr. Ting's on the PowerPC as a Forth engine, Guy Grotke's "Subversive Forth," on using Forth to write tools in a non-Forth shop for a team that otherwise had to wait for a "tools group" to develop them, and my contribution, a paper about a subroutine-threaded Forth with in-line macro substitution for the shBoom chip, originally designed by Chuck Moore and now the property of Patriot Scientific in San Diego. The history of shBoom, both legal and technical, is murky, but it's still Chuck's in spirit, a genuine Forth processor with two stacks and byte-sized machine instructions which manipulate them. Patriot is putting Java on it; I've put Forth there (it was easy—I should have won a "coals to Newcastle" award).

After the evening papers we drank wine, ate cheese, saw SwiftForth and the shBoom chip, and kicked around Dr. Ting's robot. The presence of an Asilomar employee as bartender had little dampening effect, and we scurried to an upper room afterwards as usual, though I understand that session broke up long before dawn.

One year ago, at this time, the morning after Thanksgiving, I was driving north through Big Sur, stopping for coffee at Nepenthe, thinking about what was about to happen. This year I already know.

The first Saturday morning session had perhaps the two best paper titles of the conference: Charles Shattuck and Bob Nash's "Chucking Forth," and Peter Midnight's "How I got caught in a tree by two Russians but eventually found a ladder." Nash, a good storyteller, described a project for which he and Shattuck wrote assembly language in a Forth-like way, with named macros but without a dictionary or a parameter stack, and left Shattuck a few minutes for technical details at the end. Peter Midnight's talk defies description, traveling through AVL-trees, serial programmers, and algorithmic errors to explain his title, which deserved, but did not get, a bottle of wine.

The second Saturday session had two presenters. Rob Chapman, a Canadian who's never cold in Northern California, talked about neural nets; and Elizabeth Rather talked about Forth, Inc.'s "Next Generation Forth Cross-Compilers" and their project, "Open Terminal Architecture for Europay International," perhaps the biggest Forth project currently in development.

Papers were over by Saturday lunch. The afternoon session, traditionally fragmented into working groups, departed from tradition to be, for the most part, a working group of the whole, stepping through the list of discussion topics together. It was all largely civil, no heated discussions of blocks vs. files, so that someone complained "There's no fire in this group anymore." We discussed the requirements of a modern Forth environment, bowing towards the reality that, to be used widely, it has to be Windows hosted, and has to make use of modern tools like version control. There were some relatively abstract discussions of reconfigurable hardware and Forth architecture, and concrete discussions of motion control and team software development. This last topic was particularly interesting, since the image of a Forth programmer is of a lone horse heading for the barn, but most of the people in the group claimed to have worked on Forth teams of more than three programmers. The lone horse tendency was alive, though, in resistance to code reviews,. as though professionals have nothing to learn from, or teach to, each other.

The Saturday evening impromptu talks are always a highlight, ranging from the silly to the serious. Often, the most useful bits come out of these talks, half-formed or apparently small ideas not ready for or deemed worthy of a full conference paper. It's also the time we get to hear from Wil Baden's alter egos (B.W. Daniels this year), and to hear what Chuck Moore has been doing in his spare time ("color Forth"). Someone said my presentation of Wil Baden's portable implementation of my word RETRY provoked some heat, but I didn't feel it. And one prize-winner came out of these talks: Kevin Bennet, I think the youngest attendee, got a bottle of wine for "The Most Outstanding Real-World Result" for a discussion of using Forth to control kilns in a paintbrush handle plant in Oregon.

At the wine-and-cheese party, Dr. Ting played excerpts from a CD of a performance of Eternal Sorrow, a Chinese poem by Po Chu-i which Dr. Ting set to music by Franz Schubert. Most of us sat rapt. Dr. Ting was not the singer on the CD, but he did entertain a few of us later with a Chinese tongue-twister about hunting stone-lions with a bow-and-arrow, which involved a lot of hissing. That night there may have been no after-party session: at least, many of us regulars went to bed early.

A year ago, at this time on this day, I was checking in to FORML, getting my notebook and nametag and Wil Baden's opening bear hug.

The Sunday morning session is for closing remarks, awards, thanks all around, a panel discussion, and plans for next year. Bob Reiling thanked Guy Kelly for chairing this year's session; Dr. Ting talked about his Win32Forth manual; Chuck talked about iTV; John Hart had a few things to say about a minimal Forth engine; Trace Carter talked about the state of FIG, which is getting healthier but still needs about twice as many members to be viable. After a break the final session was run by Guy Grotke, who made the panel discussion a round-robin, which got a little heated (someone said FORML this year was "pathetic," since there were so few papers, so little heat, so little prepared in advance), but gave us all a chance to mention the place of Forth in our lives.

The final note was Bob Reiling's discussion of plans for next year—same time, same place. It will be the twentieth FORML conference, a kind of milestone, and will fittingly be chaired by our noble editor, Marlin Ouverson. Remarkably, Dr. Ting got some to commit to titles for papers for next year. There will be brass bands and lion tamers. So you better all come.

After lunch we all went home, Thanksgiving still ahead.

Forth in Control:

Analog/Digital Converters

When controlling devices in the outside world using your parallel port, sometimes a need arises to monitor the device to see if it is responding properly. This feedback can be a digital signal that is fed directly into the computer, or an analog signal that needs to be converted into digital form so it can be understood by the computer.

An analog-to-digital converter is a device that senses a voltage difference between two input pins and converts it into an equivalent binary code at its output pins. When the Vin+ is equal to Vin- the converter will output a 00. When the Vin+ input analog voltage is equal to the sum of Vref and Vin-, the converter will produce the full scale code FF. The eight-bit converter we will be using (ADC0831) can convert an analog voltage of 0–5 volts into an eight-bit binary number between zero and 255.

The process of measuring an analog signal and then converting it into a binary number is called *sampling.* Each digital sample represents the amplitude of the signal at a specific time. The rate at which the binary number is updated is called the *sampling rate,* and the speed at which the converter can turn the analog input into a digital output is called the *conversion time* (32 microseconds for the ADC0831).

Analog signals are infinitely variable in amplitude, but the smallest step in which the digital output can vary is 19.53 mv; thus, the A/D converter cannot perfectly represent the amplitude of an analog signal. With Vref at 5.0 volts, the resolution of each step will be 5/256 = 19.53 mv for an eight-bit A/D converter.

The number of bits in an A/D converter determines the resolution of the converter. An eight-bit converter will discriminate to one part in 256, which is a precision of 0.4%. A ten-bit device gives a precision of better than 0.1%, and a 12-bit device a precision of 0.024%. For most general applications, eight-bit precision is sufficient.

In this article we will connect the ADC0831 to our parallel port and control it using Forth.

The ADC0831 has been around for a while and is an industry workhorse, which makes it easy to obtain and quite inexpensive. The digital output is *serial,* which cuts down on the number of pins on the chip (eight) and simplifies the interface to the parallel port.

Pin #1 of the chip is the chip select. Placing a logic low enables the A/D operation.

Pin #2 accepts the input signal Vin+ (0–5 volts).

Pin #3 is set to the zero-reference voltage (Vin-) which determines what value of Vin+ will produce a zero output reading.

Power (5 volts) is applied to pin #8 (Vcc) and Gnd to pin #4. Serial conversion data is sent out pin #6 with the most-significant bit first. The output data is synchronized with clock input on Pin #7.

The voltage applied to Vref Pin #5 determines what value of Vin+ will produce a full scale output reading of FF. The sum of Vin- and Vref must be ≤ 5 volts. For our circuit, Vin- (pin #3) will be 0 volts (Gnd) and Vref (pin #5) will be 5 volts (Vcc).

You can download a copy of the data sheet for the ADC0831 from National Semiconductor's Web site (http://www.nsc.com). Do a search by part number.

The A/D circuit is quite simple. A voltage regulator feeds a steady 5 volts to power the ADC0831 chip. The parallel port controls the chip directly through pins 1 and 7. A 10K potentiometer (pot) will simulate our sensor, which will produce a variable 0–5 volts into the chip. Build up the circuit as per schematic [page nine]. A 9 volt battery or an AC power adapter can be used for the power supply.

Plug the DB25 connector into your parallel printer port and power up the board. Run F-PC and, at the "ok" prompt, type `FLOAD ADC0831.SEQ`. If no errors are encountered, type `SHOW.DATA` to run the program. A simple display will appear, which will show the conversion value (0–255) and the calculated voltage (0–5 volts). With a small screwdriver, turn the pot (simulated sensor) to both extremes and watch the display change accordingly. To prove that the circuit is working, measure the input voltage with a voltmeter and verify this value with the displayed value. Hit any key to exit the program.

Source Code

ADC0831.SEQ automatically searches for an active LTP1 port and assigns the port address to the constant `#PORT`. If no active port is found, the error message "Parallel printer port not found" will be displayed.

The heart of the program is the word `CONVERSION`. It clears the value `ADresult` which will soon contain the conversion value read from the A/D chip. The chip is enabled and a dummy start pulse is sent to bring the Dout line out of its high-impedance state. The next clock pulse enables the most-significant bit to appear, which is read by `Dout?`. This continues for a total of eight bits, which are shifted left into the value `ADresult`. The chip is then disabled.

The word `SHOW.DATA` takes the value from `ADresult` and prints it to the screen, along with the calculated voltage.

This whole procedure is repeated until a key is pressed.

Sensors

Here is a brief list of other sensors you can implement with this circuit.

Direction Sensor

The pot we used to simulate a sensor could be made into a real sensor. Its circular motion could be used to measure direction in degrees. A single-turn pot could measure angles from 0–360 degrees. (Single turn pot, ETI Systems Model SP22E.)

Ken Merk • Langley, British Columbia, Canada
krem@vancouver.net

Ken Merk works for Canadian Pacific Railway. His article "Forth in Control: A Window Interface" appeared in issue XVIII.4, and his LED-based parallel-printer port interface was in XVII.2.

Linear motion sensor

A linear motion potentiometer is electrically identical to our circular motion pot. It is a three-terminal device that moves in and out to measure linear distances. Strokes from 1/2 inch to four inches are available. The output would be 0–5 volts, fully retracted to fully extended. The shaft can be attached to any moving part to give positional feedback to the computer. Spring return shafts are also available. (Linear motion pot, ETI Systems Series LW12.)

Acceleration/Deceleration Sensor

Acceleration is the change in velocity with respect to time, and is specified in feet per second squared. Sometimes it is referred to in terms of *g*s. One g is equal to the acceleration caused by earth's gravitational field, or 32 feet per second squared.

The ADXL50 accelerometer chip by Analog Devices will generate a voltage output proportional to any changes in velocity affecting it. The device sensitivity is factory trimmed to 19 mv/g, resulting in a full-scale output swing of ±0.95 volts for a ±50g applied acceleration. Its zero-g level is +1.8 volts. Together with its internal amplifier and a resistor network, the sensitivity of the chip can be changed. When changed to ±20g sensitivity, the zero-g output voltage is 2.5 volts, which will vary by 0.1 volt per g of acceleration. When subjected to the maximum value of ±20g, the output will swing ±2.0 volts from its zero-g level of 2.5 volts.

This chip is currently used in air bag deployment by the automobile industry. A more sensitive version of the ADXL50 is also available. The ADXL05 can detect up to ±5 g, making it useful for vibration detection. (ADLX50 ±50g range, ADLX05 ±5g range. Analog Devices data sheets available at http://www.analog.com.)

Magnetic Field Sensor

A magneto-resistive sensor made by Philips can be used to measure magnetic flux density. The sensor consists of four Hall-effect devices arranged in a bridge configuration. When it is placed in a magnetic field, a voltage is provided across it which varies linearly with the magnetic flux density. A current-carrying conductor (wire) is surrounded by a magnetic field whose strength is proportional to the value of the current. This sensor can be used to measure current flow simply by placing it in close proximity to the conductor. Current values could be monitored or it could be used simply to supply feedback that the device is operating. (Magneto-resistive sensor, Philips Type KMZ10B. Data sheet available at http://www.semiconductors.philips.com.)

Temperature Sensors

Temperature sensors LM34 and LM35 made by National Semiconductor are simple three-terminal devices. The output voltage changes 10 millivolts for every degree of change in temperature. If the temperature is 71 degrees, the output will be 0.71 volts. The LM34 output corresponds to degrees Fahrenheit, the LM35 to degrees Celsius.

In this case, if you set your Vref voltage on the ADC0831 to 2.25 volts instead of 5 volts, a 0–2.25 volt input will give a 0–255 conversion value. This way, a conversion value of 75 will correspond to 75 degrees Fahrenheit or Celsius, depending on the sensor used. (LM34 or LM35, data sheets available at http://www.nsc.com.)

Light Sensor

A simple photo-cell in series with a resistor to form a voltage divider circuit is all that is needed to measure light intensity. If the junction point of the two devices is fed into the ADC0831, a varying voltage will be present that varies with the light intensity. Applications such as intrusion alarms, event counting on production lines, or automatic dusk-to-dawn lighting are possible.

Pressure Transducers

Data Instruments makes a whole line of pressure transducers, three-terminal active devices that will output a voltage proportional to the input air pressure. Their Model SA series is encased in a steel housing and can measure up to 5000 psi. (Information available at http://info.datainstruments.com/di/welcome.html.)

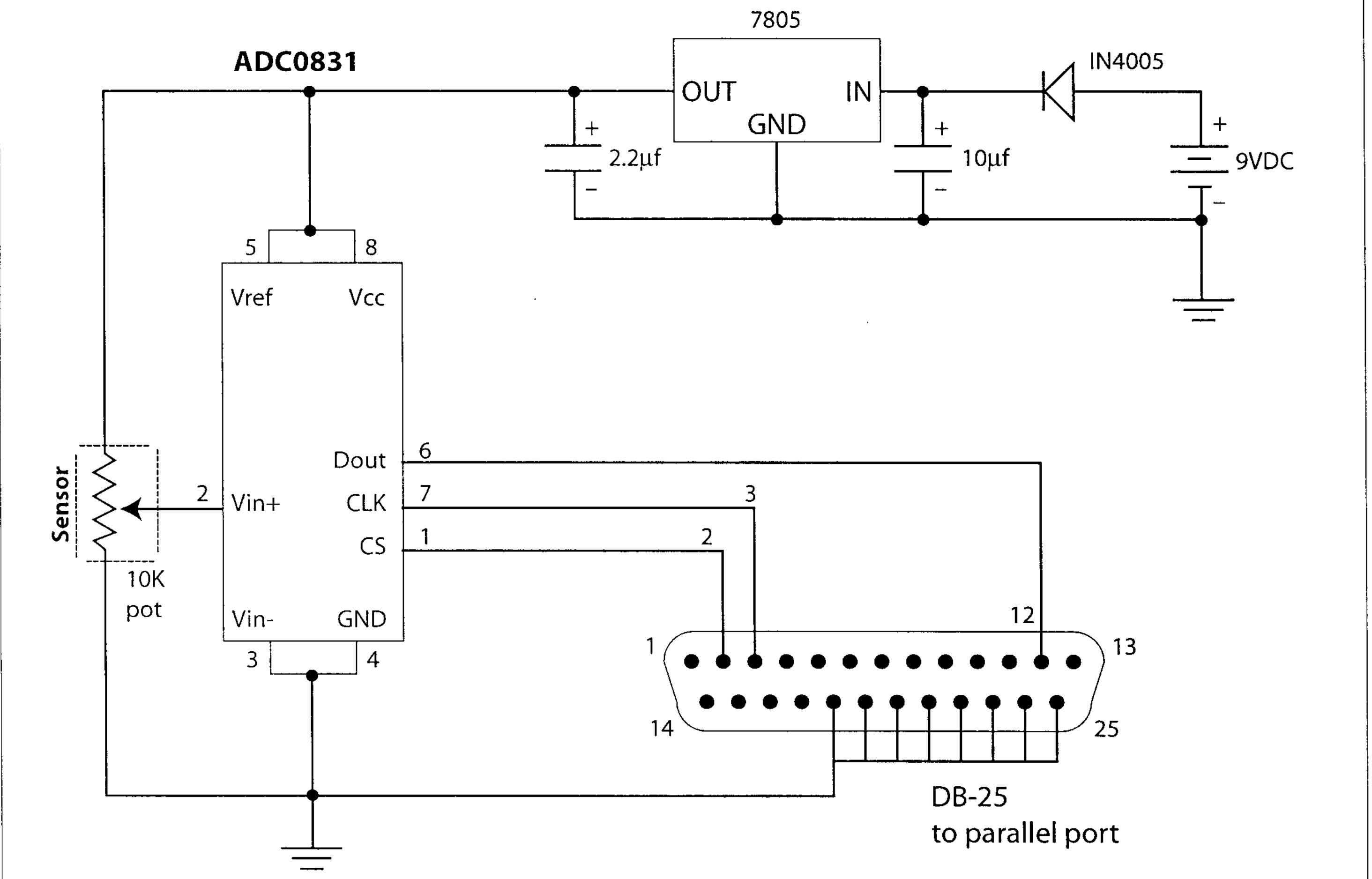
Going Further

Another useful feature of the ADC0831 is that it can operate ratiometrically. Here is an example of how this works.

Let's say you have a sensor that has a span, or dynamic voltage output range, of 0.5–3.5 volts instead of 0–5 volts. Our results would not contain the full eight-bit resolution, because of the limited voltage span (three volts). We can feed 0.5 volts to Vin- to absorb the offset, and set Vref to 3.5 volts. The ADC0831 will now encode the Vin+ signal from 0.5–3.5 volts with the 0.5 volts input corresponding to zero and the 3.5 volts input to full scale (FF). The full eight bits of resolution is now applied over this reduced input voltage range, without the need for op-amps or other scaling techniques.

Note: The most-significant bit contributes one half of the full scale voltage (2.5 volts), the next-significant bit contributes one-quarter, and so on. If you add all these eight weighted voltages that correspond to a conversion value of 255, you would find the total to be 4.9802 volts instead of 5.00 volts. Only an infinite number of bits would be able to have their corresponding voltages add up to 5.00 volts.

If a need arises for a higher performance A/D converter, look into products by Maxim—they have many varieties from which to choose. One example is the MAX-187 serial A/D converter. It comes in an eight-pin mini-DIP similar to our ADC0831, but offers 12-bit conversions. The software could be easily changed to shift in 12 bits instead of eight. (Maxim data sheets available at http://www.maxim-ic.com.)



```
\ ADC0831.SEQ                                                  Ken Merk  Aug/97
\ F-PC

\              Forth Code to Interface ADC0831 A/D Converter
\              *********************************************

   DECIMAL

     $0040 $0008 @L                        \ Look for active LPT1 port
              0= #IF                       \ If no port found then abort

              CLS
              23 8 AT .( Parallel printer port not found.)
              CLOSE QUIT

                 #ENDIF

   $0040 $0008 @L CONSTANT #PORT            \ Find port addr for printer card
                                            \ assign to constant #PORT

  0   VALUE    ADresult               \ converted data reg
  1   CONSTANT CS                     \ assign weighting to CS and CLK
  2   CONSTANT CLK

  code bset   ( b  #port -- )        \ will SET each bit in #port that matches
  pop dx                             \ every high bit in byte b.
  pop bx
```

```
in  ax, dx
or al, bx
out dx, al
next
end-code


code breset  ( b  #port -- )      \ will RESET each bit in #port that
pop dx                            \ matches every high bit in byte b.
pop bx
not bx
in  ax, dx
and al, bx
out dx, al
next
end-code


: HIGH       ( b  -- )      #PORT bset    ;   \ turn ON output
: LOW        ( b  -- )      #PORT breset  ;   \ turn OFF output
: WRITE      ( b --  )      #PORT PC!     ;   \ write byte to data port


: Dout?            32 #PORT 1+ pc@ and 0<> ;     \ READ Dout bit

: PORT.INIT        01 WRITE    ;                 \ CS=1  CLK=0

: PULSE.CLK        CLK HIGH                      \ CLK=1
                   NOOP                          \ delay
                   CLK LOW                       \ CLK=0
                   NOOP ;                        \ delay

: Format            0 <# # # # # ASCII . HOLD # #> TYPE SPACE ;


: CONVERSION           PORT.INIT               \ CS=1  CLK=0
                       0 =: ADresult           \ Clear ADresult
                       CS LOW                  \ Enable ADC0831 chip
                       PULSE.CLK               \ Send start clock pulse
                8 0 DO
                            PULSE.CLK          \ Pulse CLK
                            Dout?              \ Read Dout bit
                            ADresult 2*        \ Shift left ADresult
                            !> ADresult        \ Store into ADresult
                      IF INCR> ADresult THEN   \ IF Dout=1 INCR ADresult
                   LOOP                        \ do for 8 bits
                         CS HIGH ;             \ Disable ADC0831 chip


: SHOW.DATA         CLS                          \ Clear screen
                    cursor-off                   \ turn off cursor
            10 12 at ." A/D data -------- > "    \ Print text to screen
            10 13 at ." Voltage  -------- > "
            BEGIN CONVERSION                     \ Do a conversion
              ADresult DUP                       \ Put ADresult on stack
              30 12 at .                         \ Print VALUE to screen
              30 13 at 1953 UM* 10 UM/MOD
              SWAP 5 + 10 / +  Format KEY?       \ Print VOLTAGE to screen
            UNTIL                                \ Repeat until key-press
                  cursor-on cr ;                 \ turn on cursor
```

MicroFont: 4x5

1 Synopsis

I was working with a 128×64 graphics display and needed (mother of all inventions) some fonts to display text. I had one font which was 8×16, from an old project, but this gave me only four rows of 16 characters. The display came with a smaller font, 8×8, which gave me eight rows of 16 characters, but the set was incomplete since it only contained numbers and upper-case letters. Also, it was a bunch of hex codes and commas—difficult to visualize and extend. So I set out to define a font set which would be readable, but very small at the same time; mostly, it had to be easy to modify and build. The result is presented in this article. It is a 4×5 font I call MicroFont. It contains all printable ASCII characters, is quite readable, and the code satisfies the requirement of ease of use and design. It allows for ten lines of 25 characters on the tiny graphics display.

2 Details

Each character is four bits wide and five bits tall. This forces a few compromises when designing characters such as % or #. The four bits across fit nicely into a nibble, so each character is five nibbles. For processing efficiency, the five nibbles are fit into three bytes. So each character is three bytes long and, since there are 94 characters, the font takes up 282 bytes. In theory, two characters could fit into five bytes, and the font would then be 235 bytes; but the savings might be nullified by the increase in code to deal with it.

2.1 Character Coding

The character descriptions are done with the ASCII character * (star). Other characters could be used but this best approximated a pixel.

Figure One. The character A specified with asterisks.

```
Letter A
 **
*  *
****
*  *
*  *
```

`Letter` is a word responsible for parsing the next five lines, converting them into nibbles, packing them into bytes, and comma-ing them into the dictionary.

2.2 Code

The definitions in Figure Two are used to lay down the font.

Figure Two

```
: *>NIBBLE  ( -- nibble )  0TIB  INPUT 4 BLANKS  ( input conditioning )
  INPUT-LINE 0=  IF  ." EOF!" ABORT  ENDIF
  0  INPUT  4  FOR  C@+ >R  ` * =  IF  1 OR  ENDIF  2*  R> NEXT  DROP
  2/  0 SCAN ;

: Character  ( -- )
  2  FOR  *>NIBBLE 4 SHIFT  *>NIBBLE OR C,  NEXT  *>NIBBLE 4 SHIFT C, ;

: Letter  ( -- )  Character ;

: Number  ( -- )  Character ;
```

(Figure Two continues on next page)

Figure Three

```
( Note: nibbles are rendered across and down with msbit as left )

: ASCII>MICRO  ( char -- n )  ` ~ MIN  BL -  0 MAX ;

: .NIBBLE  ( n -- )  5 column +!  2*  5
   FOR  DUP 1 AND SET-PIXEL  2/  -1 column +!  NEXT  DROP  1 row +! ;

: microEMIT  ( char -- )  ASCII>MICRO  3 *  microfont +
  3  FOR  C@+ >R  16 /MOD  2  FOR  .NIBBLE  NEXT  R>  NEXT DROP
  -6 row +!  5 column +! ;

: MICRO-FONT  ( -- )  ' microEMIT emitter !  6 line-size !  5 char-size ! ;

: TESTMICRO  ( -- )  MICRO-FONT
  BL  ` ~ OVER -  FOR  DUP FEMIT  1 +  NEXT  DROP ;
```

Rob Chapman • Edmonton, Alberta, Canada
rc@compusmart.ab.ca

Rob Chapman is a design engineer with New Micros, working on embedded and real-time systems. His research includes translation (Timbre) and mutable hardware.

`*>NIBBLE` gets the next input line and codes the stars as bits in a nibble. The input has been preconditioned, so it will only have blanks. This is done because fewer than four characters might be read from the input and, if the previous line was longer, the next line would contain remnants of it. The input parsing could be made more complicated by checking for the end of the line, but this is sufficient and it works.

The words `Character`, `Letter`, and `Number` all do the same thing and are only provided as syntactic sugar. `Character` calls `*>NIBBLE` five times to parse the five lines. This prefont code enables us to define the font set which follows. (It should be noted that the font characters are all in one column as code but for this article they have been put into columns.)

The last set of definitions following the definition of the font is used for rendering characters to the graphics screen. Before the font sets are built, a set of variables was defined to hold the characteristics for the font and the display (see Figure Three).

`row` and `column` are used to address a pixel on the display, while `SET-PIXEL` is used to set the pixel to a foreground color or background color, depending on the truth of the flag passed to it. `emitter` holds the address of the character-rendering word which will render a character in the current font. `line-size` and `char-size` hold the size of the space used to display the font vertically and horizontally.

`ASCII>MICRO` is used to translate an ASCII character to its index into the character font table, `microfont`. `.NIBBLE` is used to output a nibble (one line of the character). The reason for the `2*` is that the nibble is converted to five bits, so the right column is blank. The least-significant bit is the right-most bit, so it must be output first. This is why the column is moved over five and then back one each time. It should be noted that the upper-left corner of the display is location 0,0. `microEMIT` is the word used to render a character. It is the equivalent of `EMIT`. `MICRO-FONT` sets the MicroFont as the current font for displaying characters. `TESTMICRO` is a test word which will display the entire font set.

This is coded for Timbre, but it shouldn't be too hard to translate to other dialects. To aid in this, I'll explain a few of the words used here. ` (back-tick) returns the ASCII value of the following letter. `FOR` and `NEXT` are a looping construct similar to `0 DO` and `LOOP`. `C@+ (a -- c \ a+)` returns the byte at the current location, leaving the incremented address on the stack. `0TIB` resets the input buffer to the beginning. `INPUT` returns the address of the current place in the input buffer. `INPUT-LINE` gets the next line of input from the input source, and returns true if it succeeded. `SCAN` accepts a character and looks through the input buffer for it, moving the input pointer along. `0SCAN` just moves the input pointer to the end of the input buffer, so there is no more input to be parsed.

3 Summary

I enjoy Forth, the people of the Forth community, and exchanging ideas with people—this article is my way of saying thanks. I hope it will be useful to someone else. A few years ago, David Lindenberg presented a similar paper, "TinyFont," at the Rochester Forth Conference. It was a 3x5 font. I chose a 4x5 font to make it a little more readable.

Figure Two (continued)

```
CREATE microfont
Character blank





Character !
  *
  *
  *

  *
Character "

 * *



Character #
 **
****
 **
****
 **
Character $
 ***
* *
 **
 * *
***
Character %
 ***
***
 *
 ***
***
Character &
  *
 * *
 **
* *
 * *
Character '
   *
  *



Character (
  *
 *
 *
 *
  *
Character )
  *
   *
   *
   *
  *
Character *

 **
****
 **

Character +

  *
 ***
  *

Character ,



   *
  *
Character -


****


Character .




  *
Character /
   *
   *
  *
 *
*
Number 0
 **
* **
****
** *
 **
Number 1
  *
 **
  *
  *
  *
Number 2
***
   *
 **
*
****
Number 3
***
   *
 **
   *
***
Number 4
  *
 **
* *
****
  *
Number 5
****
*
***
   *
***
Number 6
 **
*
***
*  *
 **
Number 7
****
   *
  *
 *
*
Number 8
 **
*  *
 **
*  *
 **
Number 9
 **
*  *
 ***
   *
 **
Character :

  *

  *

Character ;

  *

  *
 *
Character <
   *
  *
 *
  *
   *
Character =

****

****

```

```
Character >
*
 *
  *
 *
*
Character ?
 **
*  *
  *

  *
Character @
 **
** *
* **
*
 ***
Letter A
 **
*  *
****
*  *
*  *
Letter B
***
*  *
***
*  *
***
Letter C
 **
*  *
*
*  *
 **
Letter D
***
*  *
*  *
*  *
***
Letter E
****
*
***
*
****
Letter F
****
*
***
*
*
Letter G
 **
*
* **
*  *
 **
Letter H
*  *
*  *
****
*  *
*  *
Letter I
 ***
  *
  *
  *
 ***
Letter J
   *
   *
   *
*  *
 **
Letter K
*  *
* *
**
* *
*  *
Letter L
*
*
*
*
****
Letter M
*  *
****
****
*  *
*  *
Letter N
*  *
** *
****
* **
*  *
Letter O
 **
*  *
*  *
*  *
 **
Letter P
***
*  *
***
*
*
Letter Q
 **
*  *
*  *
* **
 ***
Letter R
***
*  *
***
* *
*  *
Letter S
 ***
*
 **
   *
***
Letter T
 ***
  *
  *
  *
  *
Letter U
*  *
*  *
*  *
*  *
 **
Letter V
*  *
*  *
*  *
 **
 **
Letter W
*  *
*  *
****
****
*  *
Letter X
*  *
*  *
 **
*  *
*  *
Letter Y
*  *
 * *
  *
  *
  *
Letter Z
****
  *
 *
*
****
Character [
 ***
 *
 *
 *
 ***
Character \
*
*
 *
  *
   *
Character ]
 ***
   *
   *
   *
 ***
Character ^
 *
* *



Character _



****

Character `
 *
  *



Letter a

 * *
* **
* **
 * *
Letter b
*
*
***
*  *
***
Letter c

 ***
*
*
 ***
Letter d
   *
   *
 ***
*  *
 ***
Letter e

 **
****
*
 ***
Letter f
  **
 *
***
 *
 *
Letter g
 ***
*  *
 ***
   *
 ***
Letter h
*
*
***
*  *
*  *
Letter i

  *

  *
  *
Letter j
   *

   *
   *
 **
Letter k
*
* *
**
* *
*  *
Letter l
 **
  *
  *
  *
 ***
Letter m

*  *
****
****
*  *
Letter n

***
*  *
*  *
*  *
Letter o

 **
*  *
*  *
 **
Letter p

***
*  *
***
*
Letter q

 **
*  *
 ***
   *
Letter r

* **
**
*
*
Letter s

 ***
 *
  *
***
Letter t
 *
***
 *
 *
  **
Letter u

*  *
*  *
*  *
 **
Letter v

*  *
*  *
 **
 **
Letter w

*  *
****
****
 **
Letter x

*  *
 **
 **
*  *
Letter y

*  *
 ***
   *
 **
Letter z

****
  *
 *
****
Character {
  **
  *
 **
  *
  **
Character |
  *
  *

  *
  *
Character }
**
 *
 **
 *
**
Character ~

 * *
* *
```

Formula Translation

Using Operator Precedence Grammar

Rev: 97-11-04. Allow multi-line formulas.

This is an implementation of Formula Translation. It will translate Fortran-style assignments *`varname=expr`* and expressions *`expr`* to Forth.

There is just one end-user word, `LET`. The formula is terminated by `:`. End-of-line can be used in console entry with most systems.

It can be used while compiling or interpreting. It is not state-smart.

An operand between {braces} will be treated as normal Forth.

The resulting translations are the natural expansions.

If a function doesn't begin with `F` it will first look for it with `F` prefixed.

All numbers are floating point. Variables begin with a letter, continue with letters and digits, and are not followed by a left parenthesis mark. Function calls have the same form, but are followed by a left parenthesis mark.

The operators are: `+`, `-`, `*`, `/`, `**`, or `^`.

Assignments are made with `=`. Multiple arguments of a function are separated by commas.

Spaces are deleted before translation.

Variable `DEBUG` on will show code being translated.

Let me know of any problems, please.

Thanks to Julian V. Noble for inspiration and guidance. This program uses Julian's concept but not his implementation. Thanks also to Marcel Hendrix for his ideas for extending the system.

```
LET a-b-c-d:
a F@   b F@   F-   c F@   F-   d F@   F-
LET a*b-c*d:
a F@   b F@   F*   c F@   d F@   F*   F-
LET (a-b)*(c-d):
a F@   b F@   F-   c F@   d F@   F-   F*
LET x = -1:
-1.E x F!
LET x = (-b - SQRT(b*{ FDUP} -4*a*c)) / (2*a):
b F@   FNEGATE   b F@   FDUP F*   4.E a F@   F*   c F    F*   F-   FSQRT
F-
2.E a F@   F*   F/   x F!
```

Examples of Use

Operator Precedence goes through the expression, putting out operands as it reaches them, and saving operators. Operators are put out when an operator of less or equal precedence is reached. Thus, higher precedence is performed before lower precedence.

```
FVARIABLE a   FVARIABLE b   FVARIABLE c   FVARIABLE x   FVARIABLE w
: test0    CR   LET b+c:   FE.
           CR   LET b-c:   FE.
           CR   LET 10000000*(b-c)/(b+c):   FE.
;
LET b = 3:
LET c = 4:
test0
: test1    LET a = b*c-3.17e-5/TANH(w)+ABS(x):   CR   LET a: F. ;
LET w = 1.e-3:   LET x = -2.5:   CR CR   test1
FVARIABLE HALFPI
LET HALF PI = 2*ATAN(1):
LET HALF PI + { FDUP} : F.
FVARIABLE disc                                    ( Used for discriminant )
```

Wil Baden • Costa Mesa, California
wilbaden@netcom.com

Wil Baden is a professional programmer with an interest in Forth. For the source for this article, send e-mail to WilBaden@Netcom.com requesting Stretching Forth #17: Formula Translation.

```
: quadraticroot                    ( F: a b c -- r1 r2 )
    c F!  b F!  a F!                         \ Pickup coefficients.
    LET disc = SQRT(b*b-4*a*c):              \ Set discriminant.
    LET (-b+disc)/(2*a), (-b-disc)/(2*a):    \ Put values on f-stack.
;
( Solve x*x-3*x+2 )  LET quadratic root (1,-3, 2) : F. F.
( Find goldenratio ) LET MAX(quadra ticroot (1,-1,-1)) : F.
( You can also write ) 1.E -1.E -1.E quadraticroot FMAX F.
: factorial                        ( n -- )( F: -- r )
    LET w = 1:  LET x = 1:
    0 ?DO  LET w = w * x:  LET x = x + 1:  LOOP
    LET w:
;
( Another way )
: factorial                        ( n -- )( F: -- r )
   LET w = 1:  0 ?DO  LET w = w * { I 1+ S>D D>F }:  LOOP  LET w:
;
6 factorial F.  ( or ) LET factorial({ 6}): F.
```

The program uses Tool Belt words. These are included with the distributed source.

```
( Formula Translation using Operator Precedence Grammar )

VARIABLE DEBUG  0 DEBUG !  ( This is a common name. )
( `DEBUG` occurs in one place below.  Change it here and there. )

( Character Handling )

( `replace-last-char`  Replace the last character in a string. )
: replace-last-char ( str len char -- str len )
    >R  2DUP CHARS +  R> SWAP C!
;

: isdigit ( char -- flag ) [CHAR] 0 -  10 U< ;
: isalpha ( char -- flag ) BL OR  [CHAR] a -  26 U< ;
: isalnum ( char -- flag ) DUP isalpha  ORIF DUP isdigit THEN  NIP ;

: is+or-  ( char -- flag ) DUP [CHAR] + =  ORIF DUP [CHAR] - = THEN  NIP ;
: isDorE  ( char -- flag ) BL OR  [CHAR] d -  2 U< ;

( This awful-looking code walks through the syntax for a number. )
    \ [+-]?[0-9]*([.][0-9]*)?([dDeE]([-+][0-9])?[0-9]*)?
: is-number                        ( str len -- str' len' flag )
    DUP 0= ?? FAILURE

    \ [-+]                    Any sign.
    OVER C@ is+or- IF
         1 /STRING
         DUP 0= ?? FAILURE
    THEN

    \ [.]?[0-9]       Begins with digit or decimal point and digit.
    OVER C@ isdigit  ORIF  OVER C@ [CHAR] . =  THEN 0= ?? FAILURE
    OVER C@ [CHAR] . = IF
         DUP 1 = ?? FAILURE
         OVER CHAR+ C@ isdigit 0= ?? FAILURE
    THEN

    \ [0-9]*                  Any digits.
    BEGIN  OVER C@ isdigit
```

```
   WHILE  1 /STRING  DUP 0= ?? SUCCESS
   REPEAT

   \ [ .][ 0-9]*               Decimal point and any digits
   OVER C@ [ CHAR] . = IF
        1 /STRING
        BEGIN  DUP 0= ?? SUCCESS
               OVER C@ isdigit
        WHILE  1 /STRING  REPEAT
   THEN

   \ [ dDeE] (([ -+][ 0-9] )?[ 0-9]*)?  Exponent and any sign and digits.
   OVER C@ isDorE IF
        1 /STRING
        DUP 0= ?? SUCCESS
        OVER C@ is+or- IF
             1 /STRING
             DUP 0= ?? FAILURE
             OVER C@ isdigit 0= ?? FAILURE
        THEN
        \ [ 0-9]*
        BEGIN  DUP 0= ?? SUCCESS
               OVER C@ isdigit
        WHILE  1 /STRING  REPEAT
   THEN

   SUCCESS
;

( An identifier is a letter followed by letters and digits. )
: is-identifier                ( str len -- str' len' flag )
   DUP 0= ?? FAILURE

   OVER C@ isalpha 0= ?? FAILURE

   1 /STRING
   BEGIN  DUP 0= ?? SUCCESS
          OVER C@ isalnum
   WHILE  1 /STRING  REPEAT

   SUCCESS
;
( Since you can call an identifier with spaces, no special chars. )

( Op-Stack Operations. )

30 CONSTANT Op-Stack-Size
CREATE Op-Stack   Op-Stack-Size 1+ CELLS ALLOT

: op-push                      ( op -- )
     Op-Stack @ Op-Stack-Size CELLS < NOT
         ABORT" Too Many Elements -- Increase Op-Stack-Size "
     1 CELLS Op-Stack +!   Op-Stack DUP @ + !
;

: Op-Top   ( -- op ) Op-Stack DUP @ + @ ;
: Op-Pop   ( -- )    -1 CELLS Op-Stack +! ;

( Application. )
```

```
VARIABLE Parenthesis-Count

    1 CONSTANT Left-Paren
    2 CONSTANT Right-Paren
    8 CONSTANT Negation
    9 CONSTANT Function-Call
   10 CONSTANT Op-Dummy

CREATE Word-Holder  32 CHARS ALLOT

( `memorable`  Look up variable. )
: memorable                        ( str len -- )
    31 MIN  Word-Holder PLACE           ( )
    Word-Holder FIND 0= IF
        COUNT TYPE SPACE  TRUE ABORT" Not Found "
    THEN
    DROP
;

( `callable`  Look up function. )
: callable                         ( str len -- str' len' )
    OVER C@ [CHAR] F = NOT IF
        2DUP  30 MIN  DUP 1+  Word-Holder C!
              Word-Holder CHAR+  PLACE    ( . .)
              [CHAR] F  Word-Holder CHAR+  C!
              Word-Holder FIND NIP IF
                  2DROP  Word-Holder COUNT
              THEN
    THEN                                   ( str len)
;

( `translate-operation`  [I can't think of further explanation.] )
: translate-operation              ( addr len -- )
    DEBUG @ IF  2DUP TYPE SPACE  THEN
    EVALUATE
;

( `op-store`  Make assignment. )
: op-store                         ( str len -- )( F: r -- )
    2DUP memorable translate-operation
    S" F! " translate-operation
;

( `op-fetch`  Pick up variable. )
: op-fetch                         ( str len -- )( F: -- r )
    2DUP memorable translate-operation
    S" F@ " translate-operation
;

( `op-literal`  Take care of literal. )
    VARIABLE Literal-State
: op-literal                       ( str len -- )( F: -- r )
    0 Literal-State !
    Word-Holder 0 2SWAP CHARS BOUNDS ?DO
        I C@ [CHAR] . = IF  1 Literal-State !  THEN
        I C@ isDorE IF
            Literal-State @ 0= IF
               [CHAR] .  replace-last-char  1+
            THEN
            2 Literal-State !
        THEN
        I C@  replace-last-char  1+
    1 CHARS +LOOP
    Literal-State @ 0= IF
        [CHAR] .  replace-last-char  1+
    THEN
    Literal-State @ 2 = NOT IF
        [CHAR] E  replace-last-char  1+
    THEN
    translate-operation
;

( `CASE` statements are used for ease of writing and reading. )

( `op-code`  Pick up code for operator. )
: op-code                       ( str len -- str len code )
    DUP 0= IF  0
    ELSE  OVER C@
          CASE [CHAR] )  OF  2  ENDOF
               [CHAR] +  OF  3  ENDOF
               [CHAR] -  OF  4  ENDOF
               [CHAR] *  OF  5  ENDOF
               [CHAR] /  OF  6  ENDOF
               [CHAR] ^  OF  7  ENDOF
               [CHAR] ,  OF  0  ENDOF
                             DUP . EMIT
                             TRUE ABORT" Illegal Operator "
          0 ENDCASE
    THEN
;

( `operator-precedence`  Get the precedence of an operator. )
: operator-precedence           ( code -- precedence )
    CASE -1  OF -1  ENDOF    ( Bottom Mark )
          0  OF  2  ENDOF    ( Termination or Comma )
          1  OF  1  ENDOF    ( Left Paren )
          2  OF  1  ENDOF    ( Right Paren )
          3  OF  3  ENDOF    ( Plus )
          4  OF  3  ENDOF    ( Minus )
          5  OF  4  ENDOF    ( Times )
          6  OF  4  ENDOF    ( Divide )
          7  OF  5  ENDOF    ( Power )
          8  OF  3  ENDOF    ( Negation )
          9  OF  1  ENDOF    ( Function-Call )
         10  OF  0  ENDOF    ( Dummy )
         DROP    TRUE ABORT" Invalid Operation "
    0 ENDCASE
;

( Determine what to do with the operator. )
: code-operation                ( code -- )
    CASE 1  OF  0  -1 Parenthesis-Count +!  ENDOF
         2  OF  0        ENDOF
         3  OF  S" F+ "  ENDOF
         4  OF  S" F- "  ENDOF
         5  OF  S" F* "  ENDOF
         6  OF  S" F/ "  ENDOF
         7  OF  S" F** " ENDOF
         8  OF  S" FNEGATE "   ENDOF
         9  OF  Op-Pop Op-Top  Op-Pop Op-Top
```

```
                 -1 Parenthesis-Count +!
                 callable
                 ENDOF
         DROP    TRUE ABORT" Invalid Operator "
    0 ENDCASE                    ( addr k)
    ?DUP ?? translate-operation
;

( Use operator precedence to select operators. )
: apply-operators              ( str len -- str' len' )
    BEGIN  op-code                 ( str len code)
           DUP 2SWAP 2>R           ( code code)( R: str len)
           >R operator-precedence >R  ( )( R: . . . precedence)
               BEGIN  Op-Top operator-precedence R@ < NOT
               WHILE  Op-Top code-operation  Op-Pop
               REPEAT
           R> DROP R> 2R>          ( code str len)( R: )
           DUP IF  1 /STRING  THEN
           ROT                     ( str len code)
           DUP Right-Paren =
    WHILE  DROP  Op-Pop  REPEAT
    ?DUP ?? op-push                ( str' len')
;

( Pick up an operand and an operator. )
: translate-operand-operator   ( str len -- str' len' )

    ( Is it a variable or function-call? )
    2DUP is-identifier IF     ( a n a+k n-k)
         DUP ANDIF OVER C@ [CHAR] ( = THEN IF
             Op-Dummy op-push
             DUP >R 2SWAP R> -      ( a+k n-k a k)
             op-push op-push Function-Call op-push ( a+k n-k)
             1 Parenthesis-Count +!
             1 /STRING
         ELSE
             2>R  R@ - op-fetch  2R>
             apply-operators
         THEN
         EXIT
    THEN 2DROP                          ( str len)

    ( Is it a number? )
    2DUP is-number IF      ( a n a+k n-k)
         2>R  R@ - op-literal  2R>
         apply-operators
         EXIT
    THEN 2DROP                          ( str len)

    ( Is it a left paren? )
    OVER C@ [CHAR] ( = IF
         Op-Dummy op-push  Left-Paren op-push
         1 Parenthesis-Count +!
         1 /STRING
         EXIT
    THEN

    ( Is it a lonely minus sign? )
    OVER C@ [CHAR] - = IF
         Negation op-push
         1 /STRING
         EXIT
    THEN

    ( Is it a lonely plus sign? )
    OVER C@ [CHAR] + =  ANDIF DUP 1 > THEN  IF
         1 /STRING
         EXIT
    THEN

    ( Is it evaluate? )
    OVER C@ [CHAR] { = IF
        1 /STRING
        [CHAR] } split-at-char
        2SWAP 2>R  translate-operation  2R>
        DUP IF  1 /STRING  THEN
        apply-operators
        EXIT
    THEN

    ( Oops. )
    CR  TYPE  CR
    TRUE ABORT" Illegal Operand "
;

( `translate-expression`  Translate the expression. )
: translate-expression           ( str len -- )
    BEGIN  DUP WHILE
            translate-operand-operator
    REPEAT                       2DROP
    Parenthesis-Count @ ABORT" Unmatched Parens "
;

( `translate-formula`  Translate the formula. )
: translate-formula              ( str len -- )
    0 Op-Stack !  0 Parenthesis-Count !

    2DUP is-identifier
    ANDIF DUP ANDIF OVER C@ [CHAR] = = THEN THEN IF
          DUP >R 2SWAP R> - op-push op-push -1 op-push
          1 /STRING
          translate-expression
          Op-Top -1 = NOT ABORT" Invalid Expression "
          Op-Pop Op-Top  Op-Pop Op-Top  op-store
    ELSE  2DROP
          -1 op-push
          translate-expression
    THEN

    Op-Stack @ 1 CELLS = NOT ABORT" Invalid Formula "
;

    255 CONSTANT Formula-Length
    CREATE Formula   Formula-Length 1+ CHARS ALLOT

    VARIABLE Keep-Spaces

: accept-char-for-formula    ( str length char -- str length' )
    OVER Formula-Length > ABORT" Formula Length Overflow "
    CASE
    [CHAR] {  OF  [CHAR] {  replace-last-char  1+
                  Keep-Spaces ON  ENDOF
    [CHAR] }  OF  [CHAR] }  replace-last-char  1+
                  Keep-Spaces OFF ENDOF
    [CHAR] *  OF  DUP
                  ANDIF DUP 1- CHARS Formula + C@ [CHAR] * = THEN
                      IF  1- [CHAR] ^  ELSE  [CHAR] *  THEN
                  replace-last-char  1+
              ENDOF
          replace-last-char  1+
    0 ENDCASE
;

: Get-Formula    ( "multi-lines<colon>" -- addr len )
    Keep-Spaces OFF
    Formula 0                    ( str length)
    BEGIN  GET-CHAR  DUP 0< ABORT" End of File "   ( length char)
           DUP [CHAR] : = NOT
    WHILE  DUP BL > ORIF  DUP BL = Keep-Spaces @ AND THEN  IF
                          accept-char-for-formula
                    ELSE  DROP
                    THEN         ( str length)
    REPEAT DROP
;

( `LET`   Translate `varname=expr` or `expr`. )
: LET     ( "ccc<colon>" -- )( F: -- nothing or values )
          Get-Formula translate-formula
; IMMEDIATE

(
--
Wil Baden   Costa Mesa, California
).
```

The Computer Journal

Support for older systems
Hands-on hardware and software
Computing on the Small Scale
Since 1983

Subscriptions
1 year $24 - 2 years $44
All Back Issues available.

TCJ

The Computer Journal

P.O. Box 3900
Citrus Heights, CA 95611-3900
800-424-8825 / 916-722-4970
Fax: 916-722-7480
BBS: 916-722-5799

Misty Beach Forth: An Implementation in Java

This article provides an overview of Misty Beach Forth, a Forth implementation running under Java. While, at first glance, the two technologies would seem an easy fit, the Java Virtual Machine (JVM) has peculiarities that make the fit awkward. Further, in the spirit of Java's well-defined semantics, I added some design constraints of my own that made implementing Misty Beach Forth harder than it needed to be.

What, another Forth implementation?

I have known of Forth for many years, but did little with it. Why bother? Wasn't Forth just an H-P calculator on steroids? Then, in early 1996, I read *History of Programming Languages - II*, by Bergin and Gibson. The chapter on Forth hooked me and I decided it was time to learn Forth. I decided the best way to truly understand the language was to build an implementation myself, so I started coding one in C++.

I was about ten hours into this project when I realized two things: (1) the world didn't really need another low-end x86 Forth, and (2) I had been looking for a project to do in Java for a while and maybe what the world *really* needed was a Java implementation of Forth.

I scrapped the C++ implementation and began a Java implementation. The first version was a hack designed to unearth all the problems I would have. I finished this in late 1996. After finishing, I went back and did a design for the implementation I am working on now. I expect this design to hold up.

Design goals

It is misleading to speak of design goals as if I sat down and decided up front what I wanted out of Misty Beach Forth. I didn't. I started with a general idea of what I wanted, but no grand plan. Today, I am implementing Misty Beach Forth to meet the following requirements:

1. Run as a Java applet using neither processor- nor OS-specific code.
2. Comply with the ANS Forth standard
3. Run multi-threaded Forth programs
4. Implement better-defined semantics than typical Forths
5. Provide a pleasant development environment
6. Run at speeds comparable to native Forths

Not surprisingly, running at speeds comparable to native Forths turns out to be the hardest to accomplish.

Run as a Java applet using neither processor- nor OS-specific code

Part of what allows untrusted Java applets to run securely on a client machine is a multi-level security mechanism. One of the tiers of this mechanism performs a simple proof on each method in each Java class. Among other things, this proof checks that all the operations are type-safe. I want a Forth that runs with, or on top of, the JVM and that passes all the security checks so that I can run Forth programs as applets. This requirement contributes to the difficulty of implementing Forth directly on the JVM. Forth allows words to have variable stack effects; the Java security model does not.

Comply with the ANS Forth standard

This is self-explanatory.

Run multi-threaded Forth programs

One of my interests is parallel programming, and I want a Forth that allows multi-threading.

Implement better-defined semantics than typical Forths

Most current Forth implementations provide no protection against indexing off the ends of arrays and generally stomping all over memory. In a protected operating system like Unix or Windows NT, the damage is contained to the Forth program and other programs continue to execute. I want to contain the damage even further because I dislike memory corruption bugs manifesting themselves long after the offending corruption.

Provide a pleasant development environment

My programming background is primarily in Pascal, C, C++, and Java. All these languages have inexpensive implementations with IDEs, source code debuggers, profilers, etc. I want Misty Beach Forth to provide an equally pleasant and powerful development environment.

Run at speeds comparable to native Forths

Not surprisingly, this is the hardest goal to achieve, and I am unsure whether it is actually achievable. The restrictions provided by the JVM, combined with the fact that Java applets are not coded to the native instruction set of the processor they run on, are two hurdles to overcome. The largest barrier, however, is my desire to implement a Forth with more defined semantics than typical Forths. I may have to relax the precise semantic requirement to even get close to the speed I desire.

Restrictions of the JVM

Forth is stack based. Java is stack based. At first glance, implementing Forth on the JVM appears to be no more difficult than implementing Forth on other CPUs. However, unlike most other CPUs, the JVM is designed only to run Java programs, and little consideration is paid to other programming languages. In this sense, the JVM is a special-purpose CPU, much like Forth and Lisp engines. The clash between programming models comes in several flavors:

Mark Roulo
mark@mistybeach.com

Mark has programmed professionally for eight years in C and C++, and has used Java since mid-1995. He has played with Forth for the past year—implementing an interpreter instead of using the language.

Java and the JVM are strongly typed, Forth is not

Most programming languages provide typing either for variables or values. Some provide typing for both. Forth provides typing for neither, relying instead on the programmer to know how to interpret blocks of bits. Table One illustrates the differences.

This typing for both variables and values contributes to the security provided by Java applets. The Forth approach of treating memory as a sea of bits that the programmer can interpret as desired maps poorly onto the Java model. Adding to the mismatch, the JVM implements the strong typing called for in the Java language. Writing the sort of untyped programs allowed by Forth is legal, but means that the resulting program and implementation will not pass the Java applet verifier and will not run as a Java applet.

Java and the JVM have no pointer math

Forth, like C, is built on pointer math. Simple arrays in Forth are pointers that are indexed using pointer arithmetic. An example of Forth code creating an array of two elements and storing two values in it:

```
CREATE TESTARRAY 2 CELLS ALLOT
220 TESTARRAY !
340 TESTARRAY CELL+ !
```

This has a simple equivalent in C:

```
int *p = (int *) malloc(2 * sizeof(int));
*p = 220;
*(p + 1) = 340;
```

But the Java equivalent eliminates the pointer math:

```
int[] p = new int[2];
p[0] = 220;
p[1] = 340;
```

This difference is not just syntactic sugar! Arrays in Java, unlike those in C, are objects with a well-defined API. In C, this sort of rewrite improves readability, but does not change the underlying operations. In Java, the array orientation is necessary because *p* is not a pointer to memory, but instead is a reference to an array object. This can be worked around, with tremendous effort, but the resulting Java bytecodes, once again, will not pass the Java applet verifier and will not even run on all JVMs.

The JVM stack can be discontinuous

While Java and the JVM are stack based, in the sense that operations take place on a stack of operands, the JVM breaks the stack into stack frames, one frame per function call. These stack frames are all independent. To quote from *The Java Virtual Machine Specification:*

> Because the stack is never manipulated directly except to push and pop frames, it may actually be implemented as a heap, and Java frames may be heap allocated. The memory for a Java stack does not need to be contiguous.

This causes no trouble for programs written in the Java programming language, since the Java language has no way to express the notion of accessing variables in a caller's stack frame. It does, however, mean that a subroutine-threaded Forth implementation using the JVM stack is effectively impossible, and any Forth implementation will be hard-pressed to use the JVM stack as the data stack.

The Java applet security mechanism does not allow variable stack effects

To pass the applet-verification process, functions and blocks cannot have variable stack effects. The depth of the stack upon leaving a function or block must be a fixed value relative to its depth upon entering the function or block. This restriction provides yet another hindrance to using the JVM stack as the data stack for Misty Beach Forth.

Table One

Language	Typing	Example	Explanation
C	variables	`int x = 5;`	Declare a block of memory large enough to hold an integer, and treat the bits there as if they are an integer unless told otherwise, e.g., *(float)x*. Other values assigned to this variable will be converted to an integer before being assigned, e.g., *x = (int) 9.45;*.
SmallTalk	values	`x := 1.`	Declare a variable, *x*, and make it reference a new object of type Number. If you tell the runtime to treat the Number object as if it were a Set, you get an error. You can make *x* reference something else, and this something else can be a Set, e.g., *x = Set new.*.
Java	variables and values	`Integer x = new Integer(1);`	Declare a variable, *x*, and make it reference a new object of type Integer. *x* can only reference objects of type Integer or objects of child classes of type Integer. Trying to make *x* reference a Set generates an error. You cannot tell the runtime to treat *Integer* as if it were a Set without getting an error.

Forth	neither	`VARIABLE x`
		`1 ITEM !`
			Create a new variable, *x*. Then store an integer 1 value in it. It is just as legal to store float values or pointers or characters in *x*. Further, the integer stored in *x* can be treated as a pointer, a character, or a floating-point number. The Forth runtime doesn't care.

Implementation details

Since it appeared that using the JVM stack as the data stack was not possible, I decided to build a Forth implementation roughly comparable to native-mode Forths written in C. The data and return stacks are simple arrays. I don't worry about optimal register allocation.

Some implementation details were easy to decide. Java defines the underlying virtual machine as a 32-bit, two's complement, big endian machine. This is a legal ANS configuration, so that is what Misty Beach Forth uses. The next big decision was deciding what `Cell` would look like.

Several observations drove the implementation of `Cell`:

1. Most operations are on integers.
2. Pointer math is usually used for array references.
3. Referencing off the end of an array leads to undefined behavior, so I can do what I want when this happens.

The first observation led to the decision to make integers the fundamental type. This leads to slower access times for character types, but treating characters as the fundamental type leads to even slower access times for integers. Since integer operations seem more common, integer operations are given priority. Interestingly, this leads to an architecture that is very similar to a word-oriented machine.

The second and third observations led to the creation of a `Cell` that contained an integer and an array (possibly null). Misty Beach Forth cells look like this:

```
public class Cell
{
    int    intValue;
    Cell[] arrayValue;
}
```

The implementation of +, for example, adds the `intValue` element of two cells together. The `arrayValue` element is needed to deal with address operations. To see how, we'll walk through the array example from above:

```
CREATE TESTARRAY 2 CELLS ALLOT
```

Create a new dictionary entry. The dictionary entry is an array of two `Cell`s.

```
220 TESTARRAY !
```

Place 220 on the stack:

intValue	**arrayValue**
220	null

Place `TESTARRAY` on the stack:

intValue	**arrayValue**
0	reference to `TESTARRAY Cell[]`
220	null

Set 'the value of `TESTARRAY`' to 220. This executes Java code that looks like this:

```
TESTARRAY.arrayValue[ TESTARRAY.intValue
  / 4].intValue     = 220;
TESTARRAY.arrayValue[ TESTARRAY.intValue
  / 4].arrayValue = null;
Top Of Stack = Top Of Stack - 2;
```

We divide by four in the array index, because cells are four bytes wide.

```
340 TESTARRAY CELL+ !
```

Place 340 on the stack:

intValue	**arrayValue**
340	null

Place `TESTARRAY` on the stack:

intValue	**arrayValue**
0	reference to `TESTARRAY Cell[]`
340	null

Add the size of a `Cell` to the value on the top of the stack:

intValue	**arrayValue**
4	reference to `TESTARRAY Cell[ 2]`
340	null

Set 'the value of `TESTARRAY + 4`' to 340. This executes Java code that looks like this:

```
TESTARRAY.arrayValue[ TESTARRAY.intValue
  / 4].intValue     = 340;
TESTARRAY.arrayValue[ TESTARRAY.intValue
  / 4].arrayValue = null;
Top Of Stack = Top Of Stack - 2;
```

This approach works well for reasonable Forth programs. Adding two addresses, incrementing that value, then subtracting one of the original addresses produces a valid address in most native Forth implementations. In Misty Beach Forth, it may or may not produce a valid address. Hopefully, code of this nature will be rare.

The Misty Beach Forth architecture has the notion of a *Forth engine*. This engine contains the data and return stacks, is responsible for tokenizing input strings, maintains the current state (interpreting, compiling), and contains the dictionary. The engine is almost completely ignorant of the variously defined words. With some work, I expect to make the engine ignorant of all the defined words and, thus, completely decoupled from them.

The object-oriented nature of Java provides a framework for implementing the Forth words. Forth words can be thought of as objects with both data and code. Misty Beach

Forth implements each Forth word as an object of a Java class, descending from a common base class, `ForthWord`. Each word contains methods to:

- return the name of the word (e.g., +, ALLOT)
- return the definition of the word (e.g., `: SQR DUP * ;`)
- interpret the word
- run the word
- compile the word

Performance

Designing good benchmarks is difficult. I have little experience in benchmarking, and do not wish to take the time required to become good at it. Nevertheless, without some performance numbers it is impossible to discuss relative speeds. Therefore, I present the following toy benchmark results comparing Misty Beach Forth against two shareware Forths and a commercial Forth. There are numerous problems with them, so great care should be used in drawing conclusions.

All tests ran on a 90 MHz Pentium with 64 MB RAM, running Windows NT 4.0 with no patches applied. No other applications were running.

I selected the native code Forths, based on the belief that Jax4th was a simple shareware Forth (a few hundred man hours of effort), Win32Forth was a serious shareware Forth (thousands of man hours of effort), and LMI was a serious commercial Forth implementation. What I did not realize was that, while Win32Forth *is* a serious shareware Forth, it emphasizes maximum functionality instead of speed. Serious shareware Forths emphasizing speed should perform much better.

The variable operation time for Misty Beach Forth (Table Three) is much worse than the stack operation time (Table Two). This is primarily caused by the extra indirection and memory accesses introduced to provide the better semantics I want. Preliminary tests indicate that the stack operation time can be brought down to about 15 seconds (half as fast as LMI) and the variable operation time can be brought down to about 18 seconds if I relax the safety requirement and implement a few other speed optimizations. I expect to investigate this over the next few months. An environment allowing safer runtime during test and debug with the option to convert over to a faster, but more dangerous, runtime environment later might be the best mix of speed and safety.

Table Two. Speed of stack operations

```
: INNER 10000 0 DO 34 DROP LOOP ;
: OUTER 10000 0 DO INNER LOOP ;
OUTER
```

Forth implementation	Time in seconds
Jax4th 1.25	144
Misty Beach Forth V0.30 on Netscape 3.x	92
Misty Beach Forth V0.30 on Internet Explorer 3.x	67
Misty Beach Forth V0.30 with Symantec JIT 2.0.54	59
Win32Forth 3.2 build 0819	59
Misty Beach Forth V0.30 on Netscape 4.x	54
LMI Forth (no optimization)	27
LMI Forth (optimized with NCC and COMPILE:)	7

Table Three. Speed of variable operations

```
VARIABLE TEMP
: INNER 10000 0 DO 34 TEMP ! LOOP ;
: OUTER 10000 0 DO INNER LOOP ;
OUTER
```

Forth implementation	Time in seconds
Misty Beach Forth V0.30 on Netscape 3.x	203
Jax4th 1.25	186
Misty Beach Forth V0.30 with Symantec JIT 2.0.54	144
Misty Beach Forth V0.30 on Internet Explorer 3.x	133
Misty Beach Forth V0.30 on Netscape 4.x	116
Win32Forth 3.2 build 0819	68
LMI Forth (no optimization)	35
LMI Forth (optimized with NCC and COMPILE:)	3

Conclusion

I am pleasantly surprised at how fast Misty Beach Forth runs without serious performance tuning, and expect that, with sufficient work, it can eventually run within a factor of two of serious native-mode Forths. As Java Just-In-Time compiler technology improves, the performance gap may be even lower. One of the advantages that Misty Beach Forth has is that companies like Borland, Microsoft, Sun, and Symantec are pouring millions of dollars into improving their Java Just-In-Time compilers. As these improve, Misty Beach Forth's speed improves as well. Misty Beach Forth's speed has already seen vast improvement going from Netscape 3.x to Netscape 4.x, and future improvements seem certain.

Misty Beach Forth can be found at:
http://www.mistybeach.com

References

The Java Virtual Machine, Tim Lindholm and Frank Yellin. Addison-Wesley, 1997. ISBN 0-201-63452-X

SmallTalk/V Tutorial and Programming Handbook, Digitalk, Inc. 1987 (user manual, no author or ISBN number)

History of Programming Languages - II, Thomas J. Bergin, Jr. and Richard G. Gibson, Jr. Editors, Addison-Wesley, 1996. ISBN 0-201-89502-1

Factors Influencing the Use of Forth for Large Projects

Limits to Growth

Abstract

We take a look at what factors are involved in large projects and how these impact the choice of using Forth for them. Frequently the choice of languages is dictated by managerial, customer, or agency requirements that are really political and not technical decisions. For the most part, there is very little one can do to deal with these kind of directives on a project-by-project basis. In the longer view, these kinds of decisions can be influenced by education and by ensuring that there are no technical issues regarding a particular language choice. Here we are primarily concerned with the technical issues that become important when the system being created is large.

Characteristics of large projects

Large software projects are characterized by several features:

- The specification of the system is large
- The process of verifying the system is intricate
- There are several subsystems involved
- There are multiple source code modules
- Multiple programmers are working on the development
- There is *lots* of code

Dealing with these issues drives the technical decision about the choice of language for a given project.

Measuring the complexity of a software system by the number of lines of code is not very precise. The choice of the language can have a strong influence on the actual measure. For example, a semantically weak language like Fortran can require a couple hundred lines of code to achieve the same result that a stronger language like Forth, or even C, could in a few dozen lines. Nevertheless, the lines of code does give a ballpark measure of the system's complexity.

For embedded systems, the traditional selection of programming language is typically assembler for very small projects (a few hundred lines of code at most), Forth for small to medium-sized applications (a couple thousand lines), C or C++ for medium to large-scale applications (up to a few hundred thousand lines of code), and Ada for very large applications (hundreds of thousands to millions of lines of code).

What are the reasons for these choices? The answer is that, at each of these levels of complexity, more support tools are required in order to successfully implement the system, and that such tools are available for the languages being used.

Tools to support the project

In order to support the development process for large projects, several tools beyond the compiler itself are required. These tools provide direct support for the development, plus the additional information needed to produce the required support and validation documentation. Depending upon the project, these extra tools can include:

- Tags (or definition location) support
- Concordance generators
- Flow or dependency analyzers
- Coverage analyzers
- Version control
- Configuration management

From this list, we see that the typical Forth system provides only partial coverage of the required tools.

Many Forth Integrated Development Environments (IDEs) provide a utility (e.g., LOCATE) to determine the source code location of the definition of a word. The author has also made an equivalent tool called FTAGS for the popular EMACS editor, freely available via FTP or the Web [Carter]. The nice feature of FTAGS is that it does not require a GUI and can be run from the command line (the author's computer periodically runs FTAGS automatically in the background so that the tags file is always current).

A tool to determine where definitions are established and where they are used is called a *concordance generator.* These are extremely useful for projects consisting of many source files. Concordance generators have not been part of the normal Forth toolset, but they *have* been available as separate Forth programs for some time. Feierbach and Thomas [Feierbach] provide an example for Forth-79, and the author has made one available via FTP or the Web for ANS Forth [Carter].

Dependency analyzers show the calling relationships between all the definitions. In traditional Forth systems which do not allow forward references in definitions, the dependency information is implicitly present in the organization of the code itself. For some more sophisticated Forths which *do* allow forward referencing, this information is completely lost. In either case, there is the need for a tool to generate an overview summary of how each definition is related to the others within the application.

Coverage analysis tools determine what parts of the system are actually used by the application and what are not. They provide a subset of the output of a typical concordance. While this type of analysis may not seem important, provided that the application satisfies the specified requirements, there are times when it is important to be able to establish that all the code in a system is actually going to be utilized in the application. The author recently encountered the need to make this analysis for some software because of an agency requirement that the deployed system not have any code that was not actually used. This type of concern is almost certainly based upon bad experiences with the consequences of the existence of such code written in other languages, but there are two types of systems where it is of serious concern. The first is in mission-critical systems which have certification

Dr. Everett F. Carter Jr. • Monterey, California
skip@taygeta.com

Skip Carter, a scientific and software consultant, maintains www.taygeta.com. He is also the President of the Forth Interest Group. His "Forthware" column returns in the next issue.

procedures that are to be used to qualify the software. For these systems, code that is not used will complicate the certification process and may even disqualify the code. The second situation is in software systems where security is a concern. The reason for this is the following fact: the second largest causal factor for unauthorized computer intrusion is due to "extra" code inside applications (the largest factor is poor system administration). This extra code provides features that are not part of the specification; it may be placed there for legitimate reasons such as providing a special debugging mode for the application or possibly because no effort was made to find and remove unused code, or it could be a deliberate back door to the system. In any case, because it is outside the specified requirements, such code will not be tested in the validation process, so unexpected software defects can find their way into the application.

A Forth tool for coverage analysis is not a typical part of a Forth toolset. One based upon the specialization of a concordance generator is provided in the attached listing.

Version control systems provide a way to manage the changes to source code modules. These systems provide several things at one time:

- A way to archive all changes to the source code
- A mechanism to document all source code changes
- A way to manage the changing development code when there are multiple programmers

It is uncommon to encounter Forth programmers who routinely use version control. It is telling that not a single Forth IDE system (commercial or otherwise) provides either native version control or hooks to an external version control system. The fact that, generally, Forth programmers do not use version control is unfortunate. It forces Forth to be used only in smaller applications where ad hoc source code control measures are sufficient.

Configuration management can be thought of as orthogonal to version control: whereas version control handles changes and documentation on a per-source-module basis, configuration management handles the system itself documenting which individual modules are to be used as part of a particular version of the overall system. External configuration management software can be a separate tool, or it is often done as special functions within a version management system. Again, no Forth IDE provides access to configuration management.

Summary

Some of the tools that are necessary to make Forth practical for use in very large software systems exist. However, some important tools have been neglected. Probably the most important of the missing tools are the version control systems. Forth IDE systems should either provide version control internally, or provide hooks to externally supplied version control systems. Whether version control is internal to the IDE or is done outside of the IDE, programmers must use it in order to be effective when developing non-trivial systems. This will require some self-training on the part of the typical Forth programmer, but it is vital in order to remove the technical barriers to making Forth a viable choice in developing large systems.

It is fortunate that none of these issues has anything to do with the Forth *language* itself, but are issues dealing with the state of Forth *development environments* and with the attitudes of *Forth developers*. The first is fixable by creating the tools, the second by the educational process of demonstrating the benefits of using these tools.

References

[Carter]
ftp://ftp.taygeta.com/pub/Forth *or*
http://www.taygeta.com/forth.html

[Feierbach]
Feierbach, G. and P. Thomas, 1985; *Forth Tools and Applications.* Reston Publishing Company, Prentice-Hall, Reston VA. 154 pages. ISBN 0-8359-2091-7.

Listing One

```
\ enums.fth         Words to automate the construction of a list of constants.

\                   If the number of enumerations is less than the number of bits in a word,
\                   then the enumerations are successive powers of two; otherwise they are
\                   successive integers (starting with 0).  The user can override this and make
\                   the enumerations increment by one by invoking SEQUENTIAL before ENUMS:
\
\ usage examples:
\    5 enums: one two three four five       \ powers of two constants
\    sequential 3 enums: red green blue     \ successive integers

\  This is a ANS Forth program
\
\               (c) Copyright 1996, Everett F. Carter Jr.
\                   Permission is granted by the author to use this software for any application
\                   provided this copyright notice is preserved.
\
\ $Author: skip $
\ $RCSfile: enums.fth,v $
\ $Revision: 1.2 $
\ $Date: 1996/04/18 00:13:27 $
\ ==============================================================

FALSE VALUE enum_powers?
FALSE VALUE inc_by_one

: set_inc ( n -- n )
    1 CELLS 8 * OVER < IF FALSE ELSE inc_by_one INVERT THEN
                      TO enum_powers?
;

: >enum_index ( x -- y )        \ calculate the index value
    enum_powers? IF 1 SWAP LSHIFT THEN
;

\ ==============================================================

: sequential ( -- )       \ if the user invokes this then
   TRUE TO inc_by_one     \ it will override the power of 2 increments
;

: enums: ( n -- )
    set_inc
    0 DO I >enum_index CONSTANT LOOP
    FALSE TO inc_by_one            \ make the user set this each time
;
```

Listing Two

```
#! /usr/local/bin/forth
\
\  A Forth concordance generator for analyzing Forth source files

\  This is a ANS Forth program requiring:
\       1. The CORE EXTENSIONS wordset words: CASE OF ENDOF ENDCASE
\                                  TRUE FALSE TO VALUE and \
\       2. The STRING wordset words:  CMOVE and COMPARE
\       3. The FILE wordset
\       4. The MEMORY ALLOCATION wordset.
\       5. The EXCEPTION wordset.
\       6. Heterogeneous data structure words from the Forth Scientific Library
\       7. ASCII FILE I/O words from the Forth Scientific Library
\       8. The standalone version requires access to the command
\          line arguments, the PFE version is implemented here

\               (c) Copyright 1996, Everett F. Carter Jr.
\                   Permission is granted by the author to use this software for any application
\                   provided this copyright notice is preserved.
\
\ $Author: skip $
\ $RCSfile: concordance.fth,v $
\ $Revision: 1.6 $
\ $Date: 1996/04/18 00:14:45 $

\ The following is TRUE for a STANDALONE Forth script
TRUE CONSTANT STANDALONE
STANDALONE [ IF]

variable f_index   1 f_index !

: next_file ( -- c-addr u )

    f_index @ argc >= if
```

```
      0 0
   else
      f_index @ argv
       1 f_index +!
   then
;

[ELSE]

: next_file ( -- c-addr u )
   bl word count
;

[THEN]

\ =============Support code from the Forth Library=================
s" /usr/local/lib/forth/fsl-util.nofloat.f" included
s" /usr/local/lib/forth/dynmem.f"    included
s" /usr/local/lib/forth/structs.f"   included
s" /usr/local/lib/forth/fileio.f"    included
s" /usr/local/lib/forth/enums.fth"   included
\ =================================================================

\ states
4 enums: scanning indef quoted incomment

\ state flags
true     value verbose
scanning value con_state

variable fh          \ the file handle to the current file

\ statistics on the current file
variable loc         \ lines-of-code (minus comments and blanks)
variable numdefs     \ number of colon definitions
variable numcoms     \ parenthesis delimited comments
variable numlcoms    \ "line" comments, i.e. after a backslash
variable f_string    \ s" type strings
variable c_string    \ c" type strings
variable p_string    \ ." type strings
variable numchars    \ [char] characters
variable numvars     \ number of variables (and 2variables)
variable numcons     \ number of constants
variable numvals     \ number of values
variable current_line
variable file_num

\ support data structures
structure: string
    integer:  .len
    64 chars: .str
;structure

structure: ref_table                         \ table of references
    integer: .ref_next
    integer: .ref_file     \ the file name index
    integer: .ref_line
;structure

structure: con_table
    integer:               .next             \ linked list pointer
    sizeof string struct: .name              \ the name of the def
    integer:               .line_no          \ line defined at
   integer:               .file              \ file defined at
sizeof ref_table struct:  .references        \ lines referenced at
;structure

structure: file_list
   integer:               .next_file
   sizeof string struct:  .fname
;structure

\ misc variables
string current_file
string token

128 constant bufsize
create scratch_buf bufsize allot

variable head
variable last-entry
variable flist

dynamic con_table *entry
dynamic ref_table *ref
dynamic file_list *flist

\ ======================Code==================================
: $. ( 'string -- )
   2dup .len @ >r
        .str r> type
;

: init_stats ( -- )           \ initialize per file statistics
   0 loc !
   0 numdefs !
   0 numcoms !     0 numlcoms !
   0 f_string !    0 c_string !    0 p_string !
   0 numchars !    0 numvars  !    0 numcons  ! 0 numvals !
   1 current_line !
;

: set_state ( x -- )
   con_state or to con_state
;

: clear_state ( x -- )
   invert con_state and to con_state
;

: set? ( x -- t/f )
   con_state and
;

: eol ( -- )                  \ end-of-line handler for fileio words
   indef set? if 1 loc +! then
   1 current_line +!
;

: .file_num ( n -- )          \ print a particular file name; presumes that n will be legal
   dup file_num !
   flist @ swap
   1 ?do
        (struct file_list *) swap .next_file @
      loop
      (struct file_list *) swap .fname $. ."   "
;

: ref-print ( 'ref_table -- )

   2dup .ref_line @ 0= if 2drop exit then   \ no other refs
   ." ,references: "

   0          \ stack a flag to control deleting reference
              \ structures (do not want to delete the first one)
   begin
     >r
     2dup .ref_file @
     dup file_num @ <> if cr ."     " .file_num else drop then

     2dup .ref_line @ .
     2dup .ref_next @
     r> if
         >r
         ['] *ref struct!
         ['] *ref delete-struct
        r>
         else
           >r 2drop r>
        then
     dup
   while
      (struct ref_table *) swap
       -1
   repeat

   drop
;

: .entry ( 'con_table -- )

    2dup .name $.
    ."     ,definition at: "
    2dup .file @ .file_num
    2dup .line_no @ .

   .references ref-print
   cr
;

: .table ( -- )

   head @
   begin
     dup
   while
     (struct con_table *) swap
     2dup .file @
     file_num !    \ re-using file number variable to control when to print file name
     2dup .entry
     2dup .next @ >r
     ['] *entry struct!
     ['] *entry delete-struct
     r>
   repeat

   drop
;
```

```
: file-stats ( -- )
    cr
    ." File: " current_file $. cr
    ." Approximate Lines of code: " loc @ . cr
    ." number of definitions: " numdefs @ . cr
    ." number of variables: " numvars @ . cr
    ." number of values: " numvals @ . cr
    ." number of constants: " numcons @ . cr
    ." number of comments: " numcoms @ .
    ."   line comments: " numlcoms @ . cr
    ." number of quoted characters: " numchars @ . cr
    f_string @ . ." Forth strings    "
    c_string @ . ." counted strings     "
    p_string @ . ." print strings " cr cr
;

: .flist ( -- )                 \ print the entire file list
    flist @
    begin
      dup
    while
       (struct file_list *) swap
       2dup .fname $. cr
              .next_file @
    repeat
    drop
;

: follow-links ( -- addr )      \ follow links and get address of last list address

    head head @
    begin
      dup
    while
      swap drop
      (struct con_table *) over .next @
    repeat

    drop
;

: add-file ( -- )

    flist flist @
    begin
       dup
    while
       swap drop
       (struct file_list *) over .next_file @
    repeat

    drop

    ['] *flist sizeof file_list new

    current_file 2dup .len @ >r
                      .str r>

    2over .fname .str swap dup >r cmove

    2dup  .next_file 0 swap !
    2dup  .fname .len r> swap !
```

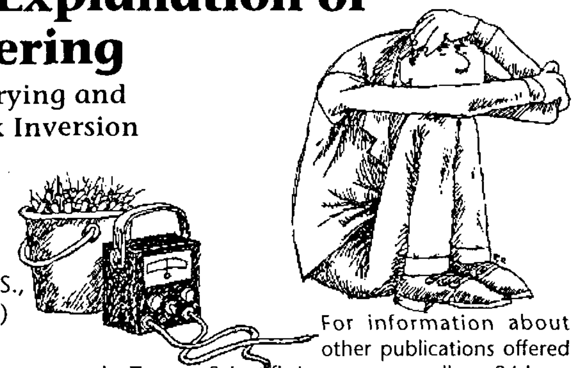
```
   swap drop swap !
;

: add-reference ( 'ref_table -- )

   2dup .ref_line @ 0=
   if 2dup .ref_file file_num @ swap !
         .ref_line current_line @ swap ! exit then

   begin
     2dup .ref_next @
     dup
   while
     >r 2drop
     (struct ref_table *) r>
   repeat

   drop
   .ref_next

   [ '] *ref sizeof ref_table new

   2dup .ref_line current_line @ swap !
   2dup .ref_next 0 swap !
   2dup .ref_file file_num @ swap !

   swap drop swap !
;

: add-definition ( -- )          \ it's a new definition
   1 numdefs +! indef to con_state

   [ '] *entry sizeof con_table new

   2dup .next 0 swap !
   2dup .file file_num @ swap !
   2dup .references 2dup .ref_next 0 swap !
                         .ref_line 0 swap !

   dup follow-links !

   2dup .line_no current_line @ swap !
   2dup .name .str fh @ get-token
   swap drop >r
        .name .len r> swap !
;

: one-char ( c -- )       \ handle special 1-character tokens
                          \ " ) : ; \ and (

   quoted set? IF [ CHAR] " = IF quoted clear_state THEN EXIT THEN

 incomment set? IF [ CHAR] ) = IF incomment clear_state THEN EXIT THEN

   CASE
       [ CHAR] : OF              \ a new definition starts
                  indef set? 0= IF add-definition THEN
               ENDOF

        [ CHAR] ; OF              \ done with a definition
                  indef set? IF scanning to con_state THEN
               ENDOF
        [CHAR] \ OF            \ skip any line comments
                scratch_buf bufsize fh @ read-line
               drop 2drop
               1 numlcoms +!
                  1 current_line +!
              ENDOF

          [CHAR] ( OF              \ start of a comment
                 incomment set_state
                  1 numcoms +!
            ENDOF

  ENDCASE
;

\ check to see if last char is a quote or end of ) comment
: test-last-char ( c-addr u -- )

    + 1- c@

        CASE
            [CHAR] " OF
                    quoted set? IF quoted clear_state THEN
                     ENDOF

             [CHAR] ) OF
                   incomment set? IF con_state clear_state THEN
                ENDOF

         ENDCASE
;

: two-char ( c-addr -- )    \ handle special 2-character tokens
                            \ S" s" C" c" ." and .(
    [CHAR] " scratch_buf 1+ c!

    DUP 2 [CHAR] S scratch_buf c! scratch_buf 2 compare
    0= IF quoted set_state DROP 1 f_string +! EXIT THEN
    DUP 2 [CHAR] s scratch_buf c! scratch_buf 2 compare
    0= IF quoted set_state DROP 1 f_string +! EXIT THEN

    DUP 2 [CHAR] C scratch_buf c! scratch_buf 2 compare
    0= IF quoted set_state DROP 1 c_string +! EXIT THEN
    DUP 2 [CHAR] c scratch_buf c! scratch_buf 2 compare
    0= IF quoted set_state DROP 1 c_string +! EXIT THEN

    DUP 2 [CHAR] . scratch_buf c! scratch_buf 2 compare
    0= IF quoted set_state DROP 1 p_string +! EXIT THEN

    DUP 2 [CHAR] ( scratch_buf 1+ c! scratch_buf 2 compare
    0= IF incomment set_state DROP 1 p_string +! EXIT THEN

    2 test-last-char

;

: is-[char]? ( c-addr -- )      \ handle [char] or [CHAR]

    DUP 6 s" [CHAR]" compare
    0= IF scratch_buf fh @ get-token 2DROP DROP
          1 numchars +! EXIT THEN

    DUP 6 s" [char]" compare
    0= IF scratch_buf fh @ get-token 2DROP DROP
          1 numchars +! EXIT THEN

    6 test-last-char
;

\ sneaky way to handle definitions
: handle-definition ( 'string -- t )    \ don't want to count constants, values, variables
     2DROP
     [CHAR] : one-char   [CHAR] ; one-char
     true
;

\ handle CONSTANTs, VARIABLEs and VALUES
: constant-or-variable? ( 'string --  t/c-addr len f)
     2dup .len @ >R
          .str r>

     2dup s" CONSTANT" compare
     0= IF 1 numcons +! handle-definition EXIT THEN

     2dup s" constant" compare
     0= IF 1 numcons +! handle-definition EXIT THEN

     2dup s" VARIABLE" compare
     0= IF  1 numvars +! handle-definition EXIT THEN

     2dup s" variable" compare
     0= IF 1 numvars +! handle-definition EXIT THEN

     2dup s" 2CONSTANT" compare
     0= IF 1 numcons +! handle-definition EXIT THEN

     2dup s" 2constant" compare
     0= IF 1 numcons +! handle-definition EXIT THEN

     2dup s" 2VARIABLE" compare
     0= IF 1 numvars +! handle-definition EXIT THEN

     2dup s" 2variable" compare
     0= IF 1 numvars +! handle-definition EXIT THEN

     2dup s" VALUE" compare
     0= IF 1 numvals +! handle-definition EXIT THEN

     2dup s" value" compare
     0= IF 1 numvals +! handle-definition EXIT THEN

     false
;

: otherwise ( 'string -- )
    constant-or-variable? if exit then

    test-last-char
;

: ?in-table ( 'string -- t/f )    \ is it in the table ?

    0 last-entry !

    head @ 0= if 2drop false exit then
    2dup .len @ >r
         .str r>

    head @
    begin
      dup
      if
         (struct con_table *) over .name
         2dup .len @ >r
              .str r>
              4 pick 4 pick compare
           0= if last-entry ! 0 else -1 then
      then
    while
      (struct con_table *) swap
      .next @
    repeat

    2drop

    last-entry @ if true else false then
;

: examine ( 'string -- )

    \ test to see if its already in the table, if so then add to the reference list

    2dup ?in-table if 2drop (struct con_table *) last-entry @
                    .references add-reference exit then

    \ not in table, check special cases to see how to handle it
    2dup .len @
    CASE
       1 OF
           2dup .str c@  one-char
       ENDOF

       2 OF
            2dup .str two-char
       ENDOF

       6 OF
           2dup .str is-[ char] ?
       ENDOF

       drop 2dup otherwise 0
    ENDCASE

    2drop
;

: <process_file> ( -- )
    verbose if
       ." processing file: " current_file $.
    then

    current_file .str current_file .len @ r/o open-file throw
    fh !

    init_stats

    1 file_num +!
    add-file
    verbose if
      cr
    then

    begin           \ loop through all the tokens in the file
      token .str fh @ get-token dup
      0 >
    while
      token .len !
      drop
      token examine
    repeat

    2drop

    fh @ close-file drop

    file-stats
;

: process_file ( -- )

    ['] <process_file> catch
    if ."   unable to open file: " current_file $. cr then
;

\ ==================The application entry point=================
: concordance ( --<file list>--  )

    cr

    ['] eol to eol-handler          \ install end-of-line handler
    0 head !   0 file_num !   0 flist !

    begin                           \ loop through all listed files
        next_file
      dup 0 >
    while
        dup current_file .len !        \ save the file name
      current_file .str swap cmove
      process_file
    repeat

    2drop

    \ .flist
    .table                          \ print concordance

    0 to eol-handler                 \ de-install EOL handler
;

STANDALONE [IF]
 concordance bye
[THEN]
```

This classic is no longer out of print!

Poor Man's Explanation of Kalman Filtering

or, How I Stopped Worrying and Learned to Love Matrix Inversion

by Roger M. du Plessis



$19.95 plus shipping and handling (2.75 for surface U.S., 4.50 for surface international)

You can order in several ways:

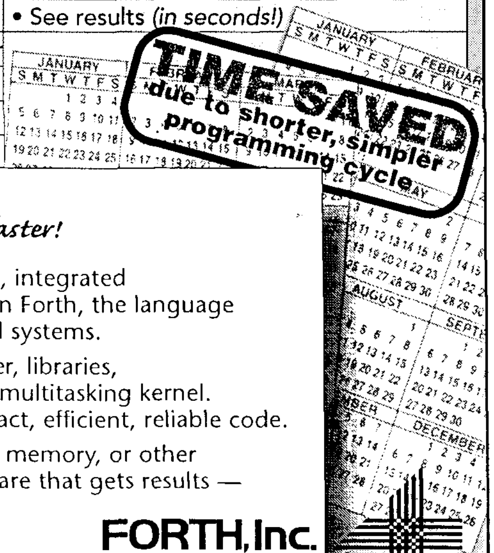
e-mail: kalman@taygeta.com
fax: 408-641-0647
voice: 408-641-0645
mail: send your check or money order in U.S. dollars to:

For information about other publications offered by Taygeta Scientific Inc., you can call our 24-hour message line at 408-641-0647. *For your convenience, we accept Master-Card and VISA.*

Taygeta Scientific Inc. • 1340 Munras Avenue, Ste. 314 • Monterey, CA 93940

GETTING RESULTS in embedded systems programming

The old way:	The SwiftX way:
• Compile	• Compile (link & download automatically)
• Link	• Execute functions immediately, by name
• Download to target	• See results *(in seconds!)*
• Set up debugger & emulator	
• Set breakpoints	
• Run	
• See results *(eventually...)*	



SwiftX™...getting results faster!

SwiftX — the new Windows-based, integrated cross-development system based on Forth, the language designed specifically for embedded systems.

Includes a cross-compiler, assembler, libraries, debugging tools, and royalty-free, multitasking kernel. SwiftX technology produces compact, efficient, reliable code.

If you have tight deadlines, limited memory, or other technical challenges, get the software that gets results — get SwiftX today!

FORTH, Inc.

111 N. Sepulveda Blvd., #300 ▪ Manhattan Beach, CA 90266
310.372.8493 ▪ 800.55.FORTH ▪ FAX 310.318.7130 ▪ www.forth.com

Finally, an *Executive Recruiter* Who Represents You the Way You Would Represent Yourself!

Contact:
Kevin Martin
Application Development Desk Specialist
Management Recruiters of Los Angeles
100 Corporate Pointe, Suite 380
Culver City, California 90230

office: 800-245-2129 (8 a.m. - 5 p.m. PST only)
office: 310-670-3040, ext. 219 (anytime)
fax: 310-670-2981
e-mail: by1989@pacificnet.net
CIS: 72020,461

- Former programmer/consult now works for you.
- I focus on finding *opportunities* rather than jobs.
- Strong conceptual skills.
- I have the contacts, now I need you.
- No cost to you.

SPONSORS & BENEFACTORS

The following are corporate sponsors and individual benefactors whose generous donations are helping, beyond the basic membership levels, to further the work of *Forth Dimensions* and the Forth Interest Group. For information about participating in this program, please contact the FIG office (office@forth.org).

Corporate Sponsors

Clarity Development, Inc. (http://www.clarity-dev.com) provides consulting, project management, systems integration, training, and seminars. We specialize in intranet applications of Object technologies, and also provide project auditing services aimed at venture capitalists who need to protect their investments. Many of our systems have employed compact Forth-like engines to implement run-time logic.

Digalog Corp. (www.digalog.com) has supplied control and instrumentation hardware and software products, systems, and services for the automotive and aerospace testing industry for over 20 years. The real-time software for these products is Forth based. Digalog has offices in Ventura CA, Detroit MI, Chicago IL, Richmond VA, and Brighton UK.

FORTH, Inc. has provided high-performance software and services for real-time applications since 1973. Today, companies in banking, aerospace, and embedded systems use our powerful Forth systems for Windows, DOS, Macs, and micro-controllers. Current developments include token-based architectures, (e.g., Open Firmware, Europay's Open Terminal Architecture), advanced cross-compilers, and industrial control systems.

The iTV Corporation is a vertically integrated computer company developing low-cost components and information appliances for the consumer marketplace. iTVc supports the Forth development community. The iTVc processor instruction set is based on Forth primitives, and most development tools, system, and application code are written in Forth.

www.theforthsource.com

Silicon Composers (web site address www.silcomp.com) sells single-board computers using the 16-bit RXT 2000 and the 32-bit SC32 Forth chips for standalone, PC plug-in, and VME-based operation. Each SBC comes with Forth development software. Our SBCs are designed for use in embedded control, data acquisition, and computation-intense control applications.

T-Recursive Technology specializes in contract development of hardware and software for embedded microprocessor systems. From concept, through hardware design, prototyping, and software implementation, "doing more with less" is our goal. We also develop tools for the embedded marketplace and, on occasion, special-purpose software where "small" and "fast" are crucial.

Taygeta Scientific Incorporated specializes in scientific software: data analysis, distributed and parallel software design, and signal processing. TSI also has expertise in embedded systems, TCP/IP protocols and custom applications, WWW and FTP services, and robotics. Taygeta Scientific Incoporated • 1340 Munras Avenue, Suite 314 • Monterey, CA 93940 • 408-641-0645, fax 408-641-0647 • http://www.taygeta.com

Individual Benefactors

Guy Grotke

John D. Hall

URLs — a selection of Web-based Forth resources

The MOPS Page
http://www.netaxs.com/~jayfar/mops.html
The Mops public-domain development system for the Macintosh with OOP capabilities like multiple inheritance and a class library supporting the Macintosh interface.

Frank Sergeant's Forth Page
http://www.eskimo.com/~pygmy/forth.html
Pygmy Forth and related files.

EE Toolbox: Software Development: FORTH Internet Resources
http://www.eg3.com/softd/forth.htm
"EG3 identifies, summarizes, and organizes the wealth of Internet information available for practical electronic design."

The Pocket Forth Repository
http://jldh449-1.intmed.mcw.edu/pf.html
A haven for programs written using Chris Heilman's Pocket Forth, a freeware Forth for the Macintosh.

AM Research, Inc., The Embedded Control Experts
http://www.amresearch.com/
AM Research has specialized in embedded control systems since 1979, and manufactures single-board computers as well as complete development systems.

Forth on the Web
http://pisa.rockefeller.edu:8080/FORTH/
A collection of links to on-line Forth resources.

Laboratory Microsystems, Inc.
http://www.cerfnet.com/~lmi/
The commercial site of LMI, with product information.

The Forth Source
http://theforthsource.com/
Mountain View Press provides educational software and hardware models of Forth with documentation for students and teachers.

The Journal of Forth Application and Research
http://www.jfar.org/
A refereed journal for the Forth community, from the Institute for Applied Forth Research.

COMSOL
http://www.forthinc.demon.co.uk/
Computer Solutions supplies tools for microprocessor designers and programmers in the U.K.

MicroProcessor Engineering, Ltd.
http://www.mpeltd.demon.co.uk/
MPE specialises in real-time and embedded systems.

The Home of the 4tH Compiler
http://www.geocities.com/SiliconValley/Bay/2334/foldtree.htm
A personal site rich in graphics and audio, as well as technical content.

Space-Related Applications of Forth
http://groucho.gsfc.nasa.gov/forth/index.html
A large table presenting space-related applications of Forth microprocessors and of the Forth programming language.

FORTH, Inc.
http://www.forth.com
Product descriptions, applications stories, links, announcements, and a history of Forth.

Forth Interest Group Home Page
http://www.forth.org/fig.html
Extensive selection of links, files, education, and a members-only section.

Forth Information on Taygeta
http://www.taygeta.com/forth.html
A selection of tools, applications, and info about the Forth Scientific Library.

Jeff Fox and Ultra Technology Inc.
http://www.dnai.com/~jfox/
Information about Forth processors.

Offete Enterprises, Inc.
http://www.dnai.com/~jfox/offete.html
Offete Enterprises has Forths for many systems and documentation about some public-domain systems.

Forth Online Resources Quick-Ref Card
http://www.complang.tuwien.ac.at/forth/forl.html
Extensive list of links to Forth enterprises and personalities.

The Forth Research Page
http://cis.paisley.ac.uk/forth/
Peter Knaggs' list of Forth resources.

Yahoo Page on Forth
http://www.yahoo.com/Computers_and_Internet/Programming_Languages/Forth/
Some of the search engine's hits on "Forth."

The Open Firmware Home Page
http://playground.sun.com/pub/1275/
Information published by the Open Firmware Working Group, provided as a free service.

American National Standard Forth Information
ftp://ftp.uu.net/vendor/minerva/uathena.htm
Courtesy of Athena Programming, Inc., working documents are posted here by direction of Technical Committee X3J14, at the discretion of the X3 Secretariat.