

ISS 0265-5195

Forthwrite

**July
2002**

Issue 117

news events people reviews projects programming



Interview with Chuck Moore

FIGUK magazine:

Special Features of kForth 2/2

Book Review

Expanding the Use of the Stack

Across the Big Teich

Vierte Dimension 1/2002

events

euroFORTH 2002	15
German FIG Annual Conference .	39

news

Forth News	2
------------------	---

reviews

Book Review “Write Your Own Programming Language using C++”	18
Across the Big Teich	31

programming

Expanding the Use of the Stack.	16
Special Features of kForth	26

people

From the ‘Net – Chuck Moore	5
Letters	41



Editorial

FIG UK continues to make a difference. Our IRC sessions attract 5 to 10 people every month and there are always visitors from outside UK. I was delighted to meet veteran Bill Ragsdale on-line in June. For the May session, we switched over to join a planned meeting on-line with Chuck Moore, the inventor of Forth. Our arrival en masse caused quite a stir among those gathered and I hope we can continue to find ways to bridge the Atlantic.

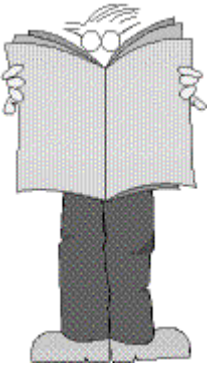
It good also to find people quoting Forthwrite on the newsgroup (for example Julian Noble quoted the JenX article recently).

We give a warm welcome to three new members; Paul Culkin from Fulham, Thierry Charlier (see letter in previous issue) and Jan Bernard van Doorn from Amsterdam.

PS. Don't forget the monthly IRC session. Our next one is Saturday 3rd August on the IRC server called "IRCNet", channel #FIGUK from 9:00pm.

Until next time, keep on Forthing,

Chris Jakeman



Forth News

Commercial Systems

iForth

iForth is a multi-platform Forth that runs on DOS, WinNT, W2K and Linux.

The new v2.0 of iForth features a completely new internal design that optimizes away most stack traffic. In benchmarks a speed increase of about 30% to 40% is measured. Compiling large projects is faster by a factor of about 1.5. Generated floating-point code outruns that produced by Microsoft's VC++ 6.0 compiler (full optimization).

4-bit Forth Microcontroller

Atmel Corporation is a worldwide leader in advanced integrated circuits and one of the elite few companies capable of integrating dense nonvolatile memory, logic and analog functions on a single chip.

Their 4-bit processor, MARC4 has a 4-bit architecture and is offered for applications such as wireless communication and reading telephone cards. Forth is used as the development environment and the CPU core is basically a Forth engine.

Non-commercial Systems

4th Website has Moved

4th is a free Forth compiler close to ANS Forth which is virtually crash-proof and can be used within C programs.

Hans Bezemer reports that the web-site has moved to

<http://www.xs4all.nl/~thebeez/4th>

Please note the Forth Primer project continues to be hosted at

<http://www.forthprimer.hothere.com/>

Help for kForth

The documentation for kforth 1.0.11 has been updated at

<http://ccreweb.org/software/kforth/kforth.htm>

The downloadable html doc package is called kforth-doc.zip and is now in synch with the on-line user's guide.

pbForth Announcements

Ralph Hempel has announced lots of developments leading to v2.1.3 which provides support for USB tower under Windows and much more reliable communications. Also support for background processing and saving an entire system. Thanks to Darin Johnson,

users can write new words in assembler right on the RCX. Source code for his H8/300 assembler is included in the distribution and tutorials are on the web-site. See

<http://www.hempeldesigngroup.com/lego/pbForth/>

The pbForth system was recently redesigned to be portable to any CPU that has a contiguous CODE/RAM space. It achieves this based on Chris Jakeman's MAF and passes the full ANS Hayes test suite.

A new tutorial has been published on playing music on the RCX. Near the end of the article, there's a description of how to hook into the 1 msec timer tick of the RCX - it's a perfect way to make a simple multi-tasker! See

<http://www.hempeldesigngroup.com/lego/pbForth/scripts/howtoRCXMusic.html>

Ralph reports around 175 active members on the pbForth mailing list.

Forth Resources

ISO Extension to 2007

Elizabeth Rather reports that the ballot run by the US ISO liaison committee oncluded on 17th April in favour of onfirming the ISO standard for a further 5 years.

ANS Forth Published Papers

In the previous Forth News, we reported that the ANS Forth of 1994 will be due for re-evaluation in 2005. Elizabeth reports that the Technical Committee (TC) “tried to reconvene in 1998-9, and did succeed in publishing papers on cross-compilers and

internationalization, but were unable to progress much farther”. See

<http://www.mpeltd.demon.co.uk/arena.htm#papers>

A new TC will be needed to take ANS Forth beyond 2005.

Chess Program

Jos Ven has published a 3D graphical chess program for Forth. The chess engine was ported by Ian Osgood and the graphics uses the OpenGL standard (which has been part of the Windows OS since Win98). The source works with Win32Forthv4.2. See

<http://home.planet.nl/~josv/>

More kForth Examples - Loans

Krishna Myneni has added to his extensive set of 57 sample Forth programs with one for calculating monthly payments on fixed interest loans. See

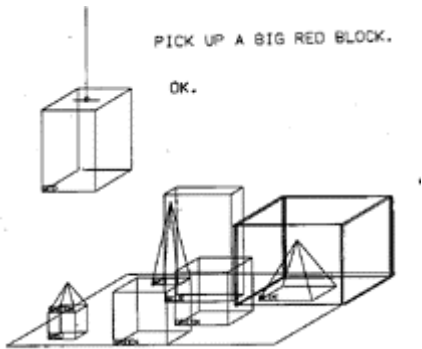
<http://ccreweb.org/software/kforth/kforth4.html>

More kForth Examples - SHRDLU

SHRDLU is a program for understanding natural language, written by Terry Winograd at the M.I.T. Artificial Intelligence Laboratory in 1968-70. SHRDLU carried on a simple dialog (via a teletype) with a user, about a small world of objects (the BLOCKS world) shown on an early display screen. See

<http://hci.stanford.edu/~winograd/shrdlu/>

Krishna Myneni has ported the “son of SHRDLU” program written by Marcel Hendrix to ANS Forth. SHRDLU provides a limited 2-D world, with



gravity, in which 4 colored boxes are placed. The user gives commands in natural language, and the program responds with the appropriate action.

A new feature of this version is that the natural language commands are entered directly at the Forth interpreter (at the ok prompt) rather than being handled by a word. The program displays "intelligence" when the user asks it to put one block over another one when either or both of the two blocks already have another block stacked on them. It also can tell the user how the blocks are positioned relative to each other in the 2-D world. See

<http://ccweb.org/software/kforth/kforth4.html>

More kForth Examples - xyPlot

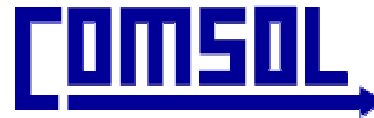
xyplot is an example of a C++ GUI application that contains a Forth environment (kForth) embedded in it. Users may extend the application by writing Forth programs that can be loaded and executed by the application. Functions may be added to the application menus, and the Forth environment can also access some C++ functions.

The latest release of xyplot uses the latest kForth environment and is available for Linux at

<http://ccweb.org/software/xyplot/xyplot.html>

Using TCP/IP to Link Microprocessors

Chris Stephens of Computer Solutions (Comsol) offers a 28-page guide to



using TCP/IP in small-memory systems, download from

<http://www.computer-solutions.co.uk>

Comsol provide several TCP/IP packages including one that fits into less than 5K of ROM.

Dynamic Strings Package

David Williams has updated his free package to v0.6.26 (When does it get to v1.0 - Ed?).

These words are intended to work with, not replace, ANS Forth string words, which act on strings represented by address, length pairs on the data stack. ANS Forth strings are especially good for analysis and parsing of strings and substrings, while dynamic strings are especially good for putting pieces of strings together and keeping them available while they're needed, then reclaiming their memory when they're not.

<http://feynman.physics.lsa.umich.edu/~williams/dstrings.html>

From the 'Net

Charles Moore (Chuck), the inventor of Forth, has always focussed on his developing his ideas rather than promoting them. He attended euroForth 2001 and was the subject of a "Slashdot" interview reported in our Nov 2001 issue. More recently, he participated in a public IRC interview hosted by James Benoit-Robey, aka futhin. The chat session was held on channel #Forth on the server irc.openprojects.net on 5th May with about 40 people attending. An edited version follows but a more complete log has been posted by Jeff Fox at <http://www.ultratechnology.com/chatlog.htm>

The monthly #FIGUK IRC session met on the same evening and switched over to #Forth to join in. Chuck's responses are in bold type. (Note: He did not have prior notice of the questions.)

Howdy. Good crowd

<futhin> I have collected a number of questions. I'm not sure if we should do a standard interview.. or a free for all. What do you think, Chuck?

Just ask questions or express opinions.

<futhin> Question from "jim": I know you're both (Chuck and Jeff Fox) involved in creating chips, and have probably gone thru several versions of forth chips... could you give us an idea of where you are in terms of stability and production?

None in production; design stable

<thefox> I haven't done a prototype run since 98 and the only MISC production run was MuP21 in 1994 by Dr. Ting.

<futhin> Question from "goshawk": How did you come to the conclusion that Forth was too complex, and that sourceless programming was your next move?

Maybe by reading the Forth Standard. There are "megaforths" that try to do everything, just like Windows or Unix. But sourceless code is a dead end. Self-limiting.

colorForth seeks the absolute minimum of overhead. Published colorForth is overly complex. colorForth in colorForth will be simple.

<futhin> Question from "AlephNull": Have you ever considered writing a book on Forth or computing?

I lack the patience to write a book. I'll let Jeff.

<futhin> Question from "futhin": What do you think about FOR NEXT? Is FOR NEXT more efficient & simple? What about the conflict with the NEXT word used in the inner interpreter in some Forths?

FOR NEXT is much simpler than DO LOOP, especially for hardware implementation. Implementation words such as the other NEXT should be

invisible. FOR NEXT runs thru loops backwards. Which is perfectly fine, once you're used to it

<futhin> Question from "wtanksley": Have you looked at backtracking? Have you played games like that with the return stack?

Yes, return stack is a valuable tool. One use I made of it was to implement infix notation with precedence operators. On the other hand, I don't like CATCH .. THROW. Errors should be impossible. Or resolved immediately.

<futhin> Question from "Fare": What do you think of high-level strongly-typed variants of forth, such as POP-11, HP RPL, Postscript?

**Typing is a crutch for poor programmers. It's an obstacle for good ones
Strong typing merely creates errors so that they can be detected
I have no objections to Forth-like languages, or any languages. Just don't make me use them.**

<futhin> Question from "kc5tja": In your ColorForth environment, you mention on your webpage that code is re-compiled on the fly, as needed. Does this happen "in place" and retroactively? Consider your RDY word in your IDE driver example code. If we were to change it somehow, and have it re-compiled, will that retroactively affect other words which utilize it?

No. Words must be defined before they're used. Original Forth, circa 1968, did provide retroactive re-definition. That means re-interpreting text at execution time. Far too expensive. But you can always recompile the application whenever you change a definition it uses. Compile time is really zero.

<futhin> "words must be defined before they are used" do you discourage the use of deferred words ?

Yes. But there are some situations when you can't avoid them. colorForth has several deferred words in the kernel. That's a flaw in the kernel design.

<futhin> Question from "jim": About catch/throw)... You don't think that in large systems, it would be nice for higher-level routines to have the option to handle lower-level exceptional conditions?

That's a hard one. There shouldn't be errors, but if a server fails to respond, something must be done. Must the application anticipate all such problems, or can the system somehow cope?

What I do is to mix the low and high-level code into an integrated whole.

Modify the low-level code as necessary for the application.

The notion of levels of code, as in communication protocols, is wrong.

There needn't be so much code to make it necessary.

<futhin> Question from "kc5tja": Are there purely technological reasons why you dislike CATCH/THROW, versus philosophical reasons?

No. It's a neat solution for the perceived problem, but when you mess with the return stack, you can create problems. Robust, reliable code is simple.

<futhin> Question from "GilbertBSD": What axioms inspired the early design of Forth?

The data stack came from the Burroughs 5500. Once I learned to use the stack, everything else followed. The stack provides name-less temporary storage. When you have to invent names, imagination fails; hence the endless hyphenated words of C.

Likewise in my chip design, most signals are unnamed, contrary to VHDL and the like that require them.

<futhin> Question from "onetom": What is your name convention for structure members/field? Whats up with nested structures?

I don't do it that way. Forth, Inc had data-base structures. The higher level names set default values. That is, a file name set the current file, a record number set the current record, a field name accesses the current record in the current file. At no time did you need to concatenate the names, though you needed to keep them distinct.

<thefox> I would add that as part of the MachineForth training I would cover the concept of using the auto-incrementing instructions as much as possible in such code.

"I don't do it that way."

<futhin> Question from "onetom": How do u avoid name clashes between fieldnames in different structures?

We were designing the database so the NAME field was in the same position in all files. If that's not possible, you need distinct names, but only in applications that must be resident simultaneously. That is, those that use multiple files. When you recompile applications as needed, this is pretty much avoided.

<futhin> Question from "futhin": "Now that you've come up with colorforth and experimented with new ways to code Forth, have you discovered any useful things that can be applied to machineForth for a better MachineForth?"

colorforth is a clearer description of MachineForth for one thing. New ideas always appear, but colorForth makes it easy to do work-arounds. For example, IF is a perennial problem. Should it pop the stack? Maybe yes, maybe no. And that's a hard change to mask. But it's simpler to decode instructions if it doesn't, so that's how it is so far.

<futhin> Question from "i440r": IF and DUP-IF ?

Having multiple words could help, but it makes the language more complex, harder to learn and doesn't address the underlying hardware issue. I could say more.

<futhin> Question from "i440r": Do you think portability is important - or does Forth's ease/speed of development negate the need for it

Portability is not important. Portability is not possible. Real applications are closely coupled to hardware. Change the platform and all the code changes. If it didn't, you wouldn't have changed the platform. To abstract the problem from the hardware requires massive software like Windows. That's a permanent tax on all applications to save some one-time programming.

Programmers should object to job-elimination concepts. Of course, jobs are actually multiplied to deal with the hyper-complex abstraction and modern hardware has computers in the displays and disks. They've already made many interfaces portable. How many layers of portability are needed?

<futhin> Question from "jim": In another question, it was mentioned you are experimenting or otherwise dealing with something called "sourceless programming"... could you elaborate a bit?

I spent several years writing sourceless code. This was my first version of OKAD, for chip design. By this I mean, editing the hex machine code into memory and saving to disk. If, as I expected, code could be reused the actual machine code would be manageable. But it grew without limit, eventually becoming unmanageable. And there was another gotcha. I'd have numbers embedded in the code, without any documentation as to how they were computed. colorForth embeds expressions and compiles the result as a literal, but the trace of what that number means, remains.

<futhin> Question from "tcn": Do you have romantic notions of fixing the internet and everything, bringing simplicity to the masses? What would it be like?

Of course. I see a Forth Markup Language (FML) supplementing HTML that defines a subset of users that can read it. FML is like colorForth - words with tags. More compact, more efficient, more flexible. Eliminates the need for Java. TCP/IP can't be changed, but it can be lived with.

<futhin> Question from "goshawk": Do you see your ideas gaining more acceptance and being more fully exploited in your lifetime? If not, what is stopping that from becoming a reality?

Ideas are memes. They evolve unpredictably. Memes insinuate themselves untraceably. My ideas have influence, probably as much as they deserve. Those that achieve prominence are the result of fads. Consider the family of languages Fortran, Algol, PLI, Pascal, C. They're all the same. The currently popular one is random choice.

No. I don't expect to become accepted; I'll just keep exploring.

<futhin> Question from "rob_ert": Do you think Forth should be used as a "general purpose language", for everyday software, or does it belong among embedded devices and other specialised systems?

Forth is the best language for all purposes because it mimics natural language; defining new words in terms of old ones. Hiding information on the stacks makes it easy for normal humans to customize their computer.

Will it happen? Give me \$100M and I'll compete with Gates.

<futhin> Question from "goshawk": Do you believe the open source "movement" had any negative impacts on the acceptance and/or practice of Forth?

Forth was open source before open source became popular. I don't see a negative impact. Any positive impact? I see a decrease in the writing of software and an increase in attention paid to integrating software.

DOD boasts about how much code can be ported from the F22 aircraft to the F35. Maybe 50% of 6M. So nobody writes from scratch, not even Linux drivers. Forth is the last bastion of DIY. Open source hasn't hurt or helped that.

"Forth is the last bastion of DIY."

<futhin> Question from "jim": Gates is presently competing with Stallman and Torvalds... if you were at that level, is it Gates that you would be competing with? Do you regret that Forth came out into open source?

I wouldn't compete on a PC platform. With the 25x chip, it's a whole new ballgame. No, I think Forth source is a goldmine. Ideas should not be secret, should not be patentable. The more people engaged, the better the result.

<futhin> Question from "josephMoore": How do you feel about automated production of software (computer driven) and its possible role in replacing future human development thereby replacing the need for language based development?

Computers that program themselves have been a dream for decades. It hasn't happened yet. Not even a little. Even if they do, it takes a human to have the insight to change the rules. Without true AI, I see no prospect and, even with AI, there has to be a language to express the problem and solution. Not in my lifetime.

<futhin> Question from "mlg": You mentioned Forth Markup Language (FML). My practice shows that Forth does not make a good language for batch programming (unless you manage to add an interactive window). What sort of interactivity (if any) do you propose for FML?

FML, like colorForth, would let the user type (steer) while processing in the background. Batch programming is an obsolete concept? Computers are so fast, anything should happen instantly. But, searching the web... Don't know. Wait and see. But always something can be done.

<futhin> Question from “futhin”: What are your further plans for an integrated Forth hardware solution? Would you be interested, money being no concern, to realize such plans?

Absolutely. Forth on a Forthchip is an unbeatable combination. Trouble is, computers are so fast already that C is viable. Yet critics claim problems that cannot be addressed. Consider weather forecasting; at some point, chaos limits predictions. Faster supercomputers are unhelpful. Perhaps UWB is fertile ground for elaborate processing (Ultra Wideband radio at <http://www.uwb.org/faqs.html>).

<futhin> Question from “joseph”: Have you ever programmed self mutable or self replicating system components or even user level applications using Forth and if so does it offer any advantage over using assembly?

OKAD, in sourceless code, was self-modifying. I'd like to optimize the c18 computer, at a layout level. That would require mutating code. colorForth is intended to facilitate that. For example, storing (and displaying) variables in source code means they can easily be changed. But no, I've not done anything significant.

<futhin> Question from “geakazoid”: Will FML have programmable tag sets like XML? What work has been done on FML? Is it a project that is being programmed?

Absolutely. Programmable tags. As in Forth, tags defined in terms of previously defined tags, something I sorely miss in HTML. An FML tag has a distinct color. It is a word that is executed by the editor and ignored by the compiler, so it takes advantage of the run/compile/edit distinction.

FML is vaporware from my perspective. What it needs is several people who use it to communicate and thereby evolve it.

<futhin> Question from “John Peters”: Where can we participate in or see some FML activity?

FML will be on the internet when its time comes. Nothing yet.

<futhin> Question from “kc5tja”: FML has been mentioned numerous times in this discussion, but yet, no examples of what it'd look like or its structure has been given. Is this iTV proprietary information? If not, can an example be posted? The idea of Forth as a markup language intrigues me much.

iTv may have precursed the idea, but not really. When I get TCP and PPP coded, I'll work on a browser. That browser will translate HTML into FML. colorForth will interpret the FML to display pages. When FML is adequate, it can be posted directly and avoid translation. For example, the purple word P could mean the HTML tag <p> and so forth.

Simple translation. Perhaps 2x compression from Huffman coding and elimination of those interminable “<”s and “>”s and the resulting spelling errors.

<futhin> Question from “Howerd”: Is there any difference between FML and a remotely executed Forth program?

FML would be a restricted subset for security, but it's not remotely executed. You've downloaded the source and executed it locally. I started it with the first colorForth, then abandoned it to work on OKAD II. One advantage of a colorForth-enabled web would be ease of sharing code.

<futhin> Question from “GilbertBSD”: Are there concepts in other programming languages that you admire or is Forth the one true language?

<futhin> Question from “fare”: Are you familiar with high-level functional programming languages such as Lisp, ML, Haskell, POP-11, Clean ? with logic programming languages such as Prolog, Mercury, Oz?

Yes, I'm familiar with LISP, Prolog, not the others. When I was developing Forth, I knew all languages. With Forth, I've neglected them. However, show me an idea (like Prolog) and I'll implement it in Forth.

But don't suggest processing of text. That's an exhausted field. LISP was one of the inspiration for Forth. The notion that you could compute something without storing anything. I don't see new ideas in the new languages.

“I wouldn't recommend me as a role model.”

<futhin> Question from “kc5tja”: Your hardware and software ventures are decidedly "out of the box". Have they been profitable enough to live comfortably with? (I guess, in other words, how big is the market?)

They've not been profitable, but they've met expenses. iTv was the best organized attempt at profitability. At one time, I was worth \$20M (paper), but it didn't work out.

I wouldn't recommend me as a role model.

<futhin> Question from “goshawk”: What inspired you to focus on hardware, and what background did you have up until that point that allowed him to make that move?

I considered software a solved problem. All my trouble with real-time systems were with the hardware. No background. Just determination.

<futhin> Question from “kc5tja” When designing the first x21 processor (i21??), by what process did you arrive at the initial 27 opcodes for the machine? Software simulations and statistical profiling?

Seat of pants. Paring down the Forth primitives till I had a managable set. My guesses corresponded well with others profiling.

<futhin> Question from “geakazoid” What is the status of Chuck & Jeff’s work on parallel processing?

The 25x is unfunded. I could fund a prototype, but without funding, where’s the market?

<futhin> Question from “mlg”: As to parallel processing, does something already work? What approach do you use to distribute computation across processes?

I would distribute functionally. For example, one processor would accept serial input, aother would process PPP, another would process IP, another would process TCP. One would eventually do the application.

<futhin> Questiong from “shapr” Do you have any suggestions for new directions to explore in computing?

I’m exploring parallel computing with very small. very fast computers. That’s 64 word of ROM, 128 words of RAM, 2400 Mips. They can’t do much, but can do it very fast. With this, I hope to explore some hard problems.

Processor tasks are programmer-assigned. Partly dependent on unique processor capabilities. There are 25 independent instruction streams, 63,000 MIPS total, programmable. Nice programming problem.

ok to all

From the ‘Net

Everyone wants to be successful, but from time to time Forth is claimed to be a “failure”. Although discouraging, such claims provoke Forth users to justify our faith in Forth and point to the areas and circumstances where it is proving more successful than other tools.

Forth will never be a mainstream programming tool. Indeed, the market has room for only one dominant tool. However, Forth has proved its value as a personal productivity tool and has already survived far longer than other candidates like Modula 2.

This is not the first time that Forthwrite has tackled this issue. Here are extracts from some of the responses to a claim in the July publication of Wired magazine that "Forth is extinct", which they define as fewer than 1,000 users.

From Elizabeth Rather of Forth Inc.

It has been called to my attention that in a recent article you declared the Forth programming language "extinct". This will come as a great surprise to several thousand of our customers, as well as to Apple, IBM, and Sun Microsystems who use Forth as the basis for their boot/plug-and-play firmware.

To see a current Forth-based product you have only to look at your friendly FedEx courier, who wears the Forth-based Enhanced Super Tracker (package tracking device) on his belt. Or think about it when you watch DirectTV, whose uplink antennas are controlled by Forth-based devices.

Since my company is primarily marketing to developers of embedded systems, I can tell you that current uses of Forth range from factory automation (e.g. Owens-Corning Fiberglas) to smart cards (the SwiftOS card OS for the Atmel AT90SC cards). Our customer list includes all branches of the armed services, plus a number of government agencies; all US automobile mfrs plus Daimler-Benz; and many more.

From Stephen Pelc of MPE Ltd.

A few recent embedded applications using Forth and developed by our clients include:

- anaesthetic ventilator for operating theatres
- mass spectrometer controllers
- theatre lighting
- underwater seismic recorders
- vending machine controllers
- ISDN routers

From Anton Ertl:

Records show 24,308 accesses of gForth and its components in 20 months or over 1,200 accesses/month including 270/month for the help document alone.

From Neal Bridges:

There are more than a thousand users of Quartus Forth alone, and that's just on the handheld side.

F11-UK

provides everything needed in a professional-quality low-cost Forth controller board.

Use it in industrial or hobby projects to control a wide range of devices using the well-known multi-tasking Pygmy Forth.

Designed for hosting from a Windows or DOS PC, you can test your application as it runs on the F11-UK board itself. The board was developed by FIG UK members to provide an easy way to explore the world of controlled devices – a niche where Forth excels.

The kit includes both hardware and software and is supported and sold to members at a nominal profit through a private company.

Software

PC-based PygmyHC11 Forth compiler running under DOS produces code for Motorola HC11 micro-controller.

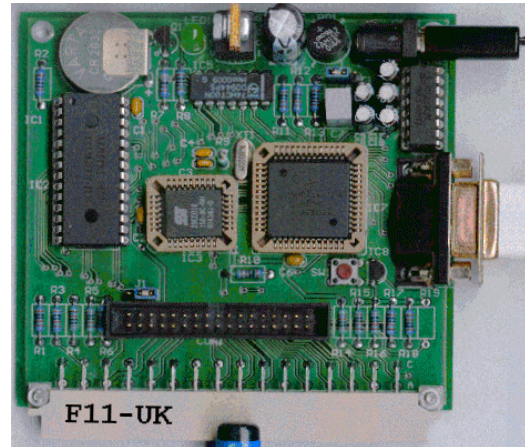
Code is downloaded via standard serial link from the PC to the FLASH memory (or RAM) on the F11-UK single board computer (SBC).

No dongle or programming adaptor of any kind is required.

Forth running on the SBC is interactive which makes debugging and testing much easier.

Multitasking and Assembly included.

The serial link can be disconnected to enable the SBC to function as a stand-alone unit.



All source code provided - 78 pages or so (unlike many commercial systems).

Around 30 pages of additional documentation is supplied including a full glossary of the 300 or so Forth words in the system.

Email mailing list for discussion and limited support.

Hardware:

Processor:

Motorola HC11 version E1 - 8 MHz (2 MHz E-Clock).

Memory:

32k x 8 FLASH
32k x 8 battery backed SRAM
512 x 8 EEPROM onboard HC11.

I/O:

20 lines plus 2 interrupts (IRQ & XIRQ).

Analogue in:

up to 8 lines using onboard 8-bit A/D.

Serial:

1. RS232, UART onboard HC11
2. Motorola SPI bus onboard HC11.

Expansion:

Via HC11 SPI serial bus using
2 or more of 20 available lines.

Timer system:

Inputs: 3 x 16-bit capture channels
Outputs: 4 x 16-bit compare channels.

PCB size: 103 x 100 mm.

Price to FIG UK members: £47.0 plus postage and packing (£2 UK, £4 overseas) plus \$25.0 (US Dollars) for registration of 80x86 Pygmy Forth with the author Frank Sergeant.

Delivery: ex-stock.
More information: jeremy.fowell@btinternet.com and 0121 440 1809

euroFORTH 2002

The 18th annual euroFORTH conference on the Forth programming environment and Forth processors is being held on Fri Sep 6th to Sun 8th at Technische Universität Wien, Vienna.

The annual conference (held in the UK every third year) moves this year to Austria. For details, see <http://www.complang.tuwien.ac.at/anton/euroforth2002>. The event begins with lunch at 12:30 on Friday and ends officially early Sunday afternoon with a “survivors party” in the evening. Special rates have been arranged at nearby hotels and the organisers are expecting the conference fee to be around 150 euros.



The conference rooms are within walking distance of Vienna city centre. (For Howerd Oakford’s report on the previous year’s conference, see Forthwrite April 2002.)

Expanding the Use of the Stack

Graham Telfer

Introduction

There was a thread on the Forth newsgroup recently about stack comments. I have always followed the traditional method and never really thought about variations.

The thread appeared just after I began reading a book about Scheme. Trying out one of the exercises and following the *design pattern* shown in the book, I decided to do the exercise in Forth.

The Area of a Doughnut

This was the exercise set. Find the area of a doughnut as the difference between the area of a large disc and the area of a small disc. Figure 1 shows the idea.

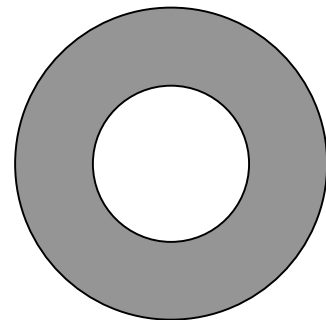


Figure 1

While doing this and following the thread about stacks, I realised that I really want the stack to do two different jobs. First I want to state the formal arguments that a word needs and secondly I want to know the actual state of the stack is immediately before and after a word is executed.

I tried to satisfy the two needs in this exercise.

Playing Around

I tried putting the formal arguments in front of a word and the stack comments after the word:

```
    formal arguments          stack comments
: ( u1,u2 --> u3)  Forth_word  ( u1,u2 --u3)  ... code ... ;
```

but quickly found this didn't work because Forth thinks that the first character following the colon is a new Forth word being defined.

Next I tried putting the Forth word first, followed by two sets of comments:

```
                formal argument    stack comments
: Forth_word ( u1,u2 --> u3)  ( u1,u2 --u3)  ... code ... ;
```

This got very clumsy and did not reflect what I felt about my needs.

The final try shows what I think works quite well. The formal arguments needed by the word precede the Forth word and the actual status of the stack follows. I also used three other devices to make things clearer.

For formal arguments I knew would be put on the stack during execution, (rather than be present at the time of execution), I put in square brackets: []. The input arguments are followed by a: --> in the formal argument part and by the standard: -- in the stack comment part. In the stack comments I put arguments that need protecting in curly brackets {}.

The Final Code

This is the finished code. The output from Result is a bit brutal since I became more interested in commenting the stack than making things pretty. Maybe though the ideas about showing both formal and stack arguments are worth developing.

```
\ Helper Words

314 Constant Pi
( u--> )      : Result ( donut_area --)
                CR ." The area of the donut is " 8 .R  ;
( u1,u2 --> u3) : FindArea
    ( outer_radius, {inner_radius}|inner_radius,{outer_area} --)
      (inner_radius,{outer_area}|outer_area,inner_area )
      Swap Dup * Pi * ;

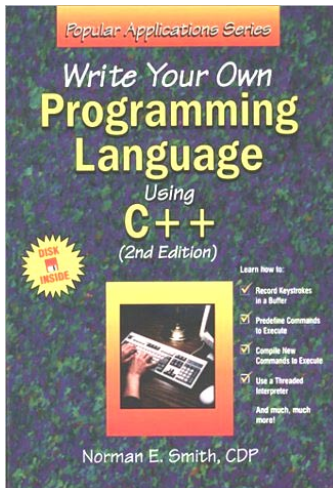
\ Main Words

( u1,[pi] --> u2) : OuterArea
    ( outer_radius,{inner_radius} --inner_radius,outer_area)
      FindArea ;

( u1,[pi] --> u2) : InnerArea
    ( inner_radius,{outer_area} -- outer_area,inner_area)
      FindArea ;

( u1,u2 -> u3)    : DonutsArea
    ( outer_area,inner_area -- donut_area)
      OuterArea InnerArea - ;

( u1,u2 -->)     : DonutArea ( outer_radius,inner_radius)
                  DonutsArea Result ;
```



fennema@gofree.indigo.ie

Book Review “Write Your Own Programming Language using C++”

Boris Fennema

New member Boris is using this book to write a Forth-inspired scripting language for use as a test driver and for interactive debugging. He writes “I was really impressed that, within 2 days after receiving the book from Amazon, I had variables, consts and words working in a fashion.”

This book is still in print (Amazon for instance) and can also be borrowed from the FIG-UK library. It comes with an 5 1/4" diskette but I have not been able to peruse this disk (yet).

**Writing Your Own Programming Language in C++
Norman E. Smith
WordWare Publishing Inc
ISBN 1-55622-264-5
March 1996**

Overview

The main aim of this little book (108 pages) is to enable a programmer to write a custom application language.

UNTIL stands for Unconventional Threaded Interpretive Language. It is *not* a Forth but borrows a number of concepts from Forth:

- Reverse Polish Notation.
- Data Stack & return stack used in loops.
- Dictionary
- interpreter
- both primitive and high-level words

The primitive words are coded in 'C'.

There is a lot of information in this book. Some of it is specific to UNTIL but, because UNTIL is close to Forth, it also gives some insight in the broader family of Threaded Interpretive Languages (TIL).

As an example, the author uses UNTIL to implement a calculating 'macro' language. His aim is for UNTIL to be portable and embedded¹ into another application written in C (or C++). The example comes with clear diagrams and in critical areas he prints small portions of C code to illustrate how UNTIL works.

There is no hidden magic in the book; it shows the strength of simple concepts (RPN, data stack, return stack) when used together. It uses a simple RPN parser as the 'outer interpreter' and the 'C' calling mechanism as the 'inner interpreter'. You can write your own TIL based on the book without reading the companion diskette (as I did since my 5 1/4" floppy drive is "somewhere" in our attic !).

UNTIL uses the indirect-threading mechanism – see below. There are faster threading models but indirect-threading is sufficient for this application and easily implemented.

Intended Readership

These are programmers who need a custom application language that is interpreted and compiled. Even though the title refers to 'C++', this is a misnomer. The publishers are probably taking advantage of the interest in C++ around that time. You could easily implement all the features in 'C' or any other language that supports function pointers, data pointers, and pointers to pointers.

An interest in languages and/or Forth would also help - given that you are reading this in Forthwrite means that criterion is probably satisfied.

Style

The author followed the Forth conventions for the dictionary headers, so the Forth Programmer's Handbook (Forth Inc.) complements this book nicely.

The topics are nicely laid out with attention to detail. Some chapters are a bit short (3 pages !) but all the essential information is there. The diagrams are especially clear and the source code is concise and well laid out.

¹ Readers interested in embedding Forth into other applications as a debugger or command language should also take a look at FICL from John Sadler at <http://ficl.sourceforge.net/ficl.html>. Unlike UNTIL, Ficl is supported, ANS Forth and available on a wide variety of platforms. Ficl includes a simple but capable object model that can wrap existing data structures.

Executive Summary

This is an useful book to have read. It can be used as a basis of an application-specific language or as the basis of a minimal Forth-like language that you can 'bundle' into another program.

It enhanced my understanding of the indirect-threading model. The code developed can also be used as a "playground" for testing ideas for enhancements quickly.

It also does a great job of showing the strengths of keeping a solution simple; this makes for an compiler/interpreter that you can explain to someone else in less than an hour !

It shows how a Forth system is an interpreter and compiler at the same time. As the book suggests, you could even rewrite the outer interpreter to use any other grammar (whether RPN or not) and the inner interpreter would still work as is. In that case the outer interpreter would shield the user from the data stack.

Boris has supplemented his book review with a description of the inner workings of UNTIL revealed by his own tools.

How does UNTIL work?

The core structures are the Data stack, Return stack and the dictionary. The Data stack is the main mechanism for argument passing. All StackDatums are signed longs in UNTIL (4 byte integers). The Return stack is used in loops but not for the call and return mechanism. For this we use the 'C' function call stack frames. All primitives are coded in 'C' and must take and place their parameters on the Data stack; i.e. they must match the prototype

```
void FooBar(void);
```

The Dictionary holds a linked list of DictHeaders. These structures contain the following information:

```
DictHeader | - nfa -- name
           |   -- length
           |   -- smudge flag
           |   -- immediate
           | - cfa -- code pointer ('C' primitive.)
           | - pfa -- additional information
           | - lfa -- link to next header.
           | - type -- type bit field (extension see next section).
```

where

- nfa = a name field address (which also records whether the word is immediate and hides the word when under compilation (smudge)).
- lfa = link field address - this links to the next entry if any in the vocabulary.
- cfa = code field address - this must point to a 'void f(void)' primitive. On invocation of any word, the 'cfa' is executed.
- pfa = parameter field address; this field is a union, and contains a pointer field or value field. It is used when defining a high-level word or when defining variables or constants.

For instance, the dictionary entry for a constant declaration looks like:

```
10 constant LIMIT
```

The nfa will contain the name (name = LIMIT, length=5), and the pfa will contain the value of the constant (10).

The cfa will point to a 'C' primitive that will place the constant value on the data stack when invoked.

The lfa will point to the next dictionary header or be null.

The important point is that any construct (constants, if's or do-loop) is represented by a dictionary header that ties a run-time function (cfa) and additional information (pfa) together when invoked or referenced.

Compilation Process

A shortened form of the compilation process for colon definitions appear below. For details, please see the book; I just want to give a flavour here.

The compilation words are ':' and ';' - these words are coded in 'C' and are "pre-loaded" into the dictionary. They are immediate, so when these tokens are encountered, the corresponding 'C' functions pointed at by their 'cfa' fields are executed instead of the dictionary entry added to the list of words under construction.

All compiler words are immediate.

The ':' moves from interpret mode to compile mode and scans in a loop for tokens separated by whitespace.

For each token it is decided whether this is a word (by looking in the dictionary); if so and if it is immediate, it is executed, else a reference to the word is compiled into the word under construction (via a DictHeader *). If it is not a recognised word, it may be a number and the conversion is attempted. If successful, a literal is build into the definition, else an error is reported.

';' stops the compilation process and moves back to interpret mode.

The new word is then stored in the dictionary under the given name. In this case the 'cfa' points at the 'C' primitive rtl_colon - the 'pfa' contains a null-terminated array of DictHeader *.

When the new word is now invoked, control is transferred to rtl_colon.

This function is shown below. It simply walks the array of DictHeader pointers and executes the 'cfa' belonging to the dictionary entry.

Since the Instruction Pointer is global, each 'cfa' can influence where the IP ends up. Branches used in loops and 'if' clauses use this to skip back and forth over the DictHeader * array. In the code below (which is derived from the book):

- gfAbort is a global flag for stack under/overflow.
- gppIP is a pointer to a DictHeader pointer – Instruction Pointer.
- gpWA is the word address pointer.
- DoStart is a routine that restarts the interpreter in case of an error.

```
rtl_colon(void) // runtime part of colon
{
    // switching context.
    DictHeader ** ppOldIP = gppIP;

    gpIP = gpWA->itsPFA.ppItsW_addr;
    gpWA = *gppIP++;

    // running the word
    while (gpWA && !gfAbort) {
        (*gpWA->itsCFA)();
        gpWA = *gppIP++;
    }

    if (gfAbort) DoStart();

    // switch context back
```

```

    gppIP = ppOldIP;
}

```

As you can see this is quite straightforward - this is also representative of the level of complexity of code in the book.

Extension - the 'see' word:

In implementing my own scripting language, the only deviations I made from the book are with respect to the storing of literal values and strings. In the declaration above, the `itsType` field is really only used when decoding words so the debugging word `see` (explained below) can print additional information on the words. It wastes some header space but is really useful when learning to use UNTIL.

As explained in the book, in a compiled word you want to load the value at run-time onto the stack. However, the value is known only at compile-time and hence needs to be saved somewhere for use at run-time.

The UNTIL language stores the value of a literal in the next dictionary slot. This works and uses space efficiently. It also means that the PFA (Parameter Field Address) can contain a non-header. However, I wanted to develop an extension word called `see` (along the lines of PC-Forth) that would allow me to see the structure of the headers laid down for a compiled word. This is very useful for debugging.

The problem was that I could only deal with dictionary headers – a value masquerading as a dictionary header caused grief since the `see` word would try to access it as an valid dictionary header address rather than the value with nasty results.

The solution was to use an entire dictionary header to store a literal. The same was done for strings that are embedded in words, eg `."`. Each dictionary header has a field to tell `see` how it needs to be accessed (this is the easiest; an alternative is to remember the previous `DictHeader cfa` field and use that information.

This works very well - the only values that can appear in a PFA list for a word are now guaranteed to be dictionary headers. It would be trivial to add a single step debugger now I have the `see` macro.

Example of Applying see

Given the following Forth code loaded into my UNTIL dialect "ENABLE":

```

: msg ." top is 5 " ;
10 constant LIMIT
: w2 LIMIT 0 do i dup 5 = if msg drop else . then loop ;

```

When `see w2` is executed, the output shows how a constant is loaded, and how `if` and `do .. loop` constructs work (branching added for clarity):

```

dissassembly of 'w2'
(1 = 'w2')
(1) (00674194) - LIMIT
(1) (00672B88) - rtl_lit
(1) (00674204) - 0 (0 decimal)
(1) (00672BC0) - rtl_do
(1) (00672F7C) - i
(1) (0067344C) - dup
(1) (00672B88) - rtl_lit
(1) (0067423C) - 5 (5 decimal)
(1) (00673174) - =
(1) (00672C68) - rtl_zbranch
(1) (00674274) - 5 (5 decimal)

(2 = 'msg')
(2) (00672CA0) - rtl_dot_quote
(2) (00674128) - top is 5 - (string)

(1) (006734BC) - drop
(1) (00672C30) - rtl_branch
(1) (006742AC) - 2 (2 decimal)
(1) (00672FB4) - .
(1) (00672BF8) - rtl_loop
(1) (006742E4) - ffffffff3 (-13 decimal)

```

The output from `see` starts each line with a level in brackets, indicating whether `see` is tracing a word in the current definition or a nested word that is called.

Note that the level for the `msg` word is 2 - this means that this is executed as a separate call to `rtl_colon` and appears as a single dictionary entry in the `pfa` list of the `w2` word. What `see` shows you is an execution trace combined with a dictionary header dump.

In other words, `msg` is not copied (in-lined) into the `w2` word, it is executed as a separate level within `w2`.

The next value is the pointer value of the dictionary entry (in hexadecimal). The remainder of the line contains details about the dictionary header itself.

As you can see:

- Loading a constant is more space efficient than loading a literal (LIMIT takes 1 DictHeader, whereas the 0 and the 5 take two).

- This can be used as an optimisation by declaring
0 constant 0
1 constant 1
etc. for a number of frequently-used integers.
- The branching targets for the `if` and the `loop` are indicated by the line drawings - the offset dictionary entry follows the branching entry.
- The `if` is created as a conditional branch, the `loop` is a direct branch which is taken as long as the index does not match the limit.
- Note that the offset for the `do loop` is 'backward' (-13 entries) whereas the `if` jumps ahead (+5).

The proof is in executing 'w2':

```

executing w2
0 - (dec 0)
0x1 - (dec 1)
0x2 - (dec 2)
0x3 - (dec 3)
0x4 - (dec 4)
top is 5
0x6 - (dec 6)
0x7 - (dec 7)
0x8 - (dec 8)
0x9 - (dec 9)

```

As you can see, it works !

What Next?

You can implement as much or as little of the Forth words you need. I plan to add words for exercising DLLs by added stubs that will take their arguments of the stack and invoke the DLL function.

Special Features of kForth

Krishna Myneni and David P. Wallace

kForth was originally written, as many Forth's are, to provide user-programming facilities within another application. However kForth includes two unusual features which are reported in this 2-part paper², which was prepared for JFAR, the Journal of Forth Applications and Research.

Krishna Myeni is a member of FIG UK living in the USA.

Dynamic Dictionary

Traditional Forth implementations use a fixed size dictionary to hold word definitions and user created data such as small tables of numbers or counted strings³. The motivation to implement a dictionary which can grow as needed may be expressed in a simple code statement:

```
create array 1024 1024 * allot
```

where we wish to allot 1 MB of space in the dictionary to hold an array of values. With a conventional static dictionary, the above code is successful only if the dictionary happens to have been allocated with sufficient space. Otherwise, the Forth system may issue a dictionary overflow error or simply crash with a segmentation fault error. Some Forth systems allow the user to resize the dictionary from within the environment. A fixed size dictionary is not desirable when kForth is used as an interpreter embedded into an application, since the useful dictionary size will depend on the application.

The ANS standard provides extension words for allocating dynamic memory, `ALLOCATE` and `FREE`, so one may argue that it is not necessary to be able to allot memory in the dictionary space for large blocks of user data. Also, having limited dictionary space to hold Forth code is usually not of practical concern. So what are the benefits of a growable dictionary? A dynamically allocated dictionary provides the following conveniences to the programmer:

- `ALLLOT` may be used without size restriction. The programmer is not burdened by determining whether or not there is sufficient space to locate a block of data in the dictionary.
- Addresses of data blocks made with the sequence, `CREATE name size ALLLOT`, do not have to be managed by the programmer. With `ALLOCATE`, the returned address must be assigned to a constant or held in a variable until the memory block is freed.

² The paper is available as in PDF format from <ftp://ccreweb.org/documents/programming/nsf-a.pdf>

³ L. Brodie, Starting Forth 2nd ed., (Prentice Hall, Englewood Cliffs, NJ, 1987)

- Memory is freed automatically upon exit from the Forth system. In contrast, memory obtained using `ALLOCATE` should be released using `FREE` when it is no longer needed. Forgetting to do so reduces the system memory available to other applications.

These features are of little value in programming embedded processors, which have stringent limits on available memory. However, for a desktop system using a modern operating system (e.g. Linux, Windows), programming is less cumbersome with these features. However, the benefits to the Forth programmer from using a dynamic dictionary come with some restrictions. The implications of using a dynamic dictionary are discussed below.

In kForth, memory for both code and data is dynamically allocated as required. In this allocation scheme, the task of providing dictionary space and assigning addresses is passed on to the operating system (OS) rather than being handled by the Forth system, and growth of the dictionary is limited only by the OS.

The dictionary itself is a vector of data structures, each containing the name of a word, the word's precedence, the code field address (CFA), and the parameter field address (PFA). In kForth, CFA is synonymous with code pointer or the ANS term, execution token. PFA is synonymous with the ANS term data field. During "compilation" of a word definition, a temporary code vector is built up. The size of this vector is unbounded. After a word definition has been compiled into code, which in kForth consists of pseudo op-codes, memory is dynamically allocated to hold the code sequence and then copied from the vector into the newly allocated block. The PFA of the new word is set to the address of the dynamically allocated block. The CFA is also set. If the word `RECURSE` was encountered during compilation, address placeholders inside the code are then replaced with the CFA.

Now we consider the behavior `ALLOT` may have in a system which implements a dynamic dictionary. First note that there is no `HERE` address in the dynamic system, since memory is not available until it is requested either through word definitions or by execution of `ALLOT`. In the traditional static dictionary Forth system, the code:

```
1024 allot
```

presumes that the programmer has access to the starting address of the memory region to be allotted, either because `CREATE` was invoked previously or because the starting address was obtained with `HERE`. Therefore, unlike its counterpart `ALLOCATE`, `ALLOT` does not return an address.

In the dynamic dictionary system, kForth, the code `1024 allot`, must dynamically allocate 1024 bytes of memory, starting at some address which is determined by the OS. This address must somehow be made available to the programmer to allow use of the memory. We must change the behavior of `ALLOT`,

but wish to do so in a way that use of ALLLOT remains as consistent as possible with traditional Forth code.

kForth imparts the following behavior to ALLLOT; the requested memory is dynamically allocated and the starting address is assigned to the PFA of the last word defined in the dictionary. In kForth ALLLOT must be used only in a CREATE name size ALLLOT sequence. The behavior of CREATE is also modified so that it sets the PFA to zero for the new dictionary entry. This allows ALLLOT to verify that it is modifying the PFA of a word created with CREATE, instead of modifying a word that is already associated with data or code. Therefore, the statement:

```
1024 allot
```

by itself produces an error in kForth:

```
VM Error(9): Allot failed --- cannot reassign pfa
```

Two other core words are not provided in kForth owing to the lack of a HERE address. These are:

- the comma operator (,)
- C,

For creating initialized cell-size or byte-size tables, alternative, albeit somewhat less elegant, methods can be used instead. For instance, instead of the simple statement:

```
create tbl 100 , 200 , 300 , 400 ,
```

to make a 4 element table initialized to values, we could write

```
: t, ( n a1 -- a2 ) 2dup ! 1 cells - nip ;  
create tbl 4 cells allot  
100 200 300 400 tbl 3 cells + t, t, t, t, drop
```

Clearly the statement using the comma operator is simpler but has the problem that an address for storing initial values into the table is not available until we use ALLLOT, in conjunction with CREATE, to allocate the region for the table. Then the address must be manipulated to move the successive initial values from the stack into the table in the proper order.

The example given above would be much easier with a suitable defining word for creating an initialized table in the absence of the comma operator. Such a word needs the starting address of the allotted memory so ?ALLLOT is provided to solve this problem. ?ALLLOT functions like ALLLOT but also returns the starting address of the region on the stack. The compatible ANS Forth definition of ?ALLLOT is:

```
: ?allot ( -- AddressAllocated) here swap allot ;
```

Using `?ALLOT`, we may create a defining word for initialized tables:

```
: table ( ... n -- )
  create dup cells ?allot
  over 1- cells + swap
  0 ?do dup >r ! r> 1 cells - loop drop ;
```

Using `table`, it becomes trivial to create an initialized table:

```
100 200 300 400 4 table tbl
```

Note that this method works in ANS Forth as well, provided the compatible definition of `?ALLOT` is used. `?ALLOT` should not be equated with the ANS word `ALLOCATE` since, in kForth, `?ALLOT` must be used only with `CREATE` and it assigns the PFA of the created word.

Two simple examples of defining words having run-time code further illustrate the use of `?ALLOT` in kForth:

```
: const ( n -- ) create 1 cells ?allot ! does> @ ;
: ptr ( a -- ) create 1 cells ?allot ! does> a@ ;
```

The word `const` is equivalent to `CONSTANT` and `ptr` allows the creation of address constants for our typed Forth system.

kForth code can be ported to a Forth system with a static dictionary merely by adding the definition of `?ALLOT` given above.

Summary

This 2-part article discusses two features of kForth which depart from the current ANS standard. Data typing and type checking arise from a desire to supplement the Forth environment's error detection capability, particularly for its use as an embedded interpreter in other applications. We have demonstrated that our limited method of type checking can catch common Forth programming mistakes, particularly those associated with the return stack. Also, type checking in our implementation is largely transparent to the programmer and requires only one additional word, `A@`.

The use of a dynamic dictionary offers convenience in making unrestricted use of the available system memory but at the cost of sacrificing the core words for compiling integer and byte constants into the dictionary: `comma (,)` and `C,.` Furthermore, `ALLOT` must be used only in conjunction with `CREATE`, and we must add the new word `?ALLOT` to allow the programming of defining words which require access to the allotted region.

It has been our experience, in using kForth both as an embedded interpreter and as a stand-alone computing environment, and for such diverse tasks as simulating microcontroller assembly code to demonstrating properties of

hydrogen atom wave-functions, that the benefits from the new features outweigh their costs.

Editor's Note: This modern ANS-style Forth should not be confused with the earlier kForth from Guy Kelly (1988, for PC). Krishna Myneni conducted an Internet search (in 1998) and found no references to the earlier work.

Krishna Myneni is a physicist and self-taught programmer who delights in devising new experiments, piecing them together out of odds and ends, and orchestrating the pieces with software. He is a long-time Forth user and proponent, much to the amusement of his colleagues.

Just for Fun

From time to time, Forth users discuss the minimum set of primitives needed by a full Forth. There are many possible sets, depending on which words you start with. The "+" word can be one of those primitives or it can be derived from other primitives as here, courtesy of Wil Baden.

```
: +                                ( x y -- x+y )
  BEGIN DUP WHILE
    2DUP AND 2* \ Calculate bit carries.
    >R XOR R>   \ Calculate bit sums.
  REPEAT DROP ;
```

Across the Big Teich

Henry Vinerts

This material was prepared for Vierte Dimension by Henry Vinerts,
and printed by permission of Forth Gesellschaft (German FIG)

FIG Silicon Valley Chapter Meeting - Mar. 2002

Spring is here and we are still "washing" Windows. Bob Smith greeted a small group of about a dozen early risers with his summary of recent changes to Tom Zimmer's Win32Forth.

Not having read Jim Lawless' original interview with Tom in English, I wonder how I would translate "Schmollecke" (page 21, VD 1/2002) back into Tom's words. Tom is leaving the "pouting corner" and going on to new things, allowing the rest of the world to play with Win32Forth, except for WinView, which is expected to re-enter as WinEd. When Tom was still in Silicon Valley, I remember that he used to quote Einstein about making things simple, but not any simpler. It is no longer possible within the paradigms created by Microsoft. There is no way that Bill Gates could have come from the same gene pool as Henry David Thoreau.

Tom also used to say that to make Forth interesting to the public, it should be presented as a game. Now it appears to me that a number of Forthers themselves, all around the world, have come to enjoy the challenge of the game of tinkering with Win32Forth.

Tom, you have inspired them all ! And Bob Smith, in designing a Windows' icon for the new WinEd editor, perhaps cunningly, has added the mark of the old pencilmaker Thoreau's simplicity by changing the stem of the fig leaf to a pencil !

It was good to see Chuck Moore again, albeit as a patient listener and not a speaker at this meeting. It's been almost 35 years since the birth of FORTH on the IBM 1130; I wonder whether Chuck might be a bit bewildered by all of FORTH's grandchildren who are in the playgrounds today.

Dr. Ting had time to state that Win32Forth was too complicated and that he was using WineForth for his Chinese character graphics project. The eForth is his, the "Win" part is from a young man in Taiwan.

John Peters had brought lunch for any helpers on his Win32Forth modification project, and a small group worked on until Mike Saari appeared with two recumbent electric bicycles and invited everyone for test rides. Definitely more fun than crashing Windows, even for old Forthers! These bicycles that Mike is building from mostly standard parts are 21-speed, reclining-type hybrids, which one can pedal or drive with the special 36-volt Heinzmann motor. With one battery, 20 to 40 miles per charge and up to 20 mph speeds are possible. With pedal assist, a daily range of 100 miles is not unusual.

I can think of many parts of the world where a \$1500 bicycle would be a luxury and I can also imagine that, given our constantly increasing traffic congestion, soon it may become a necessity for many Silicon Valley commuters. Check them out at the web pages of ElectricBetterbikes.com or SkeeterEV.com. At this point there is no Forth aboard the bikes, but it may appear, as controls become more sophisticated as electric regeneration is added.

Mike's talk drew many questions and interest from the audience, which lately has dropped in number to fewer than 20. Even at that, it seems that SVFIG is where the action is in the U.S.A., as far as an organizational character is concerned.

Hang in there, Forth-Gesellschaft e.V. and FIG UK too!

FIG Silicon Valley Chapter Meeting - Apr. 2002

Greetings, everybody!

It is Saturday evening in Garmisch, and I wish that I could join the German Forth Group with a full stein of Bavarian beer. Instead, a hearty "prosit" from California will have to do! I am sure that on this occasion of the yearly German Forth Conference I may convey the greetings and best wishes from the Silicon Valley Forth Group to you all.

Our April meeting took place early, last Saturday, and since my daughter was visiting us from Germany, I was not able to attend. I did drop in for a few minutes in the morning, however, to deliver the latest issue (#116) of Forthwrite, which had just arrived two days earlier. Dr. Ting was there, ready to give a description of his F# (F-sharp, in English), which is a 32-bit, protected mode, subroutine-threaded eForth implementation with a Windows interface.

According to the program, in the afternoon Jeff Fox was to give a short presentation on his most recent GUI desktop. Since, as I heard, he is one of the more frequent SVFIG visitors to the monthly FIG UK IRC sessions, chances are that he will exchange some information there.

Roll out the barrel and have some fun!

Cheers to all,

Henry

FIG Silicon Valley Chapter Meeting - May 2002

Hello, my friends across the Big Pond!

The Vierte Dimension arrived on May 22nd, four days after last month's SVFIG meeting. I have enjoyed reading the happy reports about the German Forth Conference in Garmisch-Partenkirchen, and it warms my heart to know that the next two conferences - in 2003 and 2004 - have already practically been assured. It should also warm Chuck Moore's heart to know that his creation has moved many people to lasting friendships and associations.

Now, for my report... I wish I knew and understood more about the Basic Stamp and Hans Eckes' Forth Stamp, so I could make some intelligent comments about Al Mitchell's presentation, which absorbed the interest of close to 20 attendees for the whole day. You see, Al has written his own AMRBASIC in GForth (the brainchild of Ertl, Paysan, and Wilke, if I remember correctly), which runs on Linux, because Al has rejected Windows for the past couple of years.

With subroutine threading and new 8051-compatible chips, Al can make his system run BASIC about 500 times faster than any of the presently available Basic Stamps can. His AMRBASIC is mostly interpretive and offers extensibility like the Forth which is underneath. Once he gets it ported to Windows, he hopes to introduce users to Forth through the medium of BASIC.

Al's website (www.amresearch.com) may not have much on this subject at this time, but some advertising is expected soon, as the system nears completion.

Since I understand English better than computer science, I'd

like to quote some of Al's comments that struck me as interesting.

"Forth appeals to people who really comprehend the problem."

"Forth is the most powerful tool possible for machine control."

"People who use Forth are outside of the ANSI (American National Standards Institute) box."

"Every application is a dialect of Forth."

The last thought reminds me of my belief that the best products seem to spring from single minds, rather than from community efforts, but it leaves me with a question as to whether the world would have been better served if Leibniz and Newton had collaborated, if Aiken had known Zuse, or even if Al Mitchell and Hans Eckes could meet over a cup of coffee sometime.

That's it for today. As you may have noticed, Dr. Ting was absent, perhaps against his better wishes, but since Cogswell College shifted our meeting date to the same Saturday that Dr. Ting's daughter was getting married (as I heard), family prevailed over the 'Verein'.

Cheers,
Henry



Alan J M Wenham
01932 786440
101745.3615@compuserve.com

Vierte Dimension 1/2002

Alan Wenham

Alan provides a look at the latest issue of the German FIG magazine. To borrow a copy or to arrange for a translation of an individual article, please call Alan.

General

Martin Bitter, who has edited the last five volumes of Vierte Dimension, has returned the task to Friederich Prinz with thanks to VD authors and readers.

New Members

Klaus Sobawa

Klaus Sobawa introduces the firm Firma Soering GmbH, who are medicine technologists, as new members of Forth Gesellschaft and reports on their computerised developments. Fred Behringer welcomes Rolf Schoene and discusses the problem of the changing age-group structure in human society and in Forth Gesellschaft.

Tower forever - under DOS

Rolf Schoene
rolf@rolf-schoene.de

Rolf shows that one can very quickly gain access to the DTR bit of the PC serial port (needed for Lego robot programming) with the help of the DOS DEBUG program and gives an assembler DEBUG script for this.

Prize award

Martin Bitter
martin.bitter@forth.ev.de

Martin presents a SWAP dragon coffee cup for solution of a riddle concerning suitable Forth code relating to DNA chain sequences.

Stack-Forth

Soeren Tiedemann

A generally philosophical view concerning Chuck Moore's machine Forth and "keep it simple". "25 Primitives" will generally suffice. "And whoever needs complicated stack operation for his/her problem has not thought it through properly or is not able to program".

Reviews

Fred Behringer
behringe@mathematik.tu-
muenchen.de

In this issue Fred Behringer reviews Forthwrite 114 and Figleaf for December 2001.

Report from the 17th Euroforth conference

Ulrich Hoffman
ulrich.E.Hoffman@gmx.de

This gathering took place at Schloss Dagstuhl from 23-25th November 2001 and has been fully reported in Forthwrite.

An interview with Tom Zimmer

Translation of the Forthwrite 114 article

From the Big Teich

Henry Vinerts

This also appears in Forthwrite.

Why is the significance of OO over-emphasised?

Andreas Klimas

This is the first part of an intended series of tutorials. Encapsulation, polymorphism and inheritance are significant aspects of object oriented programming but the first two of these are a matter of course in Forth. This leaves inheritance, which the author considers to be the key factor in OO.

FINDRAMD.COM - Assembler programming in Forth

Fred Behringer
[behringe@mathematik.tu-
muenchen.de](mailto:behringe@mathematik.tu-muenchen.de)

This is once more entry in Fred's "Column for language migrants". The program FINDRAMD.EXE is to be found on a Windows98 emergency boot diskette and it enables the assignment of a drive letter to a created RAM-disk. The hard disk on the machine may have several partitions and in order to copy necessary files to the emergency RAM-disk one must know the relevant drive letters. FINDRAMD.EXE does all this and is 6855 bytes long. FORTH-assembler can be quickly used to generate a program FINDRAMD.COM which is only 20 bytes long ! (It should in fairness be pointed out that the FINDRAMD.EXE program does include error trapping and error messages.)

MuP21/F21-Bootprocess

Soeren Tiedemann

The author discusses the installation, memory map and bus-logic, 8-bit bootmode, boot routines, and software of the above processor in what is intended to be a series of articles.

Pontifex

Friederich Prinz
Friederich.Prinz@t-online.de

Pontifex refers to the building of a bridge between Heaven and Earth. Fritz lightheartedly discusses a demo version of a program for the "virtual construction of bridges" for would-be bridge constructors available by download from <http://www.chronilogic.com> .

**German FIG
Annual Conference 2002**

Dear Readers of Forthwrite,

Our Annual Conference has been a great success, with a high total number of 37 participants, most of them members.

Anton Ertl and Klaus Schleisiek presented variations of their euroFORTH papers (see Howerd Oakford's review), and Bernd Paysan came with an interesting paper on FPGAs and Forth. Yours truly could not restrain from again reading a paper on Arithmetized Logic: A program for converting a Disjunctive Normal Form to its equivalent Multi-Linear Form.

Our Dutch guests, Willem Ouwerkerk and Albert Nijhof, travelled with Martin Bitter from the Dutch/German border to the heart of Bavaria. They presented a collection of homebrew robots, raising the interest of quite a few of us, including Martin and myself.

After many years of faithful service to Forth Gesellschaft, Thomas Beierlein has retired. The election process was full of suspense, and Bernd finally made it by one vote. So the new Board of Directors of Forth Gesellschaft reads: Ulrich Hoffman, Bernd Paysan, and Fred Behringer.

The Dragon Award 2002 was given to Hans Eckes for his "Forth Stamp" design and other achievements.



Dutch Forth Users Group

Reading Dutch is easier than you might think. And as Forth is an international language, reading Dutch code is easier still for a Forth enthusiast. Are you interested? Why not subscribe to

HCC-Forth-gebruikersgroep

For only 20 guilders a year (£6.30), we will send you 5 to 6 copies of our "fig-leaf" broadsheet 'Het Vijgeblaadje'. This includes all our activities, progress reports on software and hardware projects and news of our in-house products.

To join, contact our Chairman:

**Willem Ouwerkerk
Boulevard Heuvelink 126
6828 KW Arnhem, The Netherlands
E-Mail: w.ouwerkerk@kader.hobby.nl**

The easiest way to pay is to post a 20 Guilder note direct to Willem.

Letters

The Magazine Team are always pleased to get feedback and encouragement. The first two letters are from recipients of the FIG UK Awards.

**Dave
Pochin**

I have just received the latest Forthwrite.

What a surprise, thank you and the committee for my Award, I'm sure that there are many more deserving candidates than I.

The amount of time and effort you and the other committee members put in to support Forth is not sufficiently recognised for starters. Where would we all be without your input ?

The revision of my site is not going well just at present, nothing wrong with the content, I hope ! I am trying to improve the navigation, it looks very good, but doesn't work as well as it looks. Now, where have I seen things like that before !

Still on the Windows - Win32Forth learning curve, but the writer's block is a bit solid, and there are plenty of diversions around. After many failed attempts I am just beginning to find my way around the Forth Scientific Library, full of goodies I don't understand, but it sure beats pencil and paper for the bits I do recognise, so much to do, as always.

Regards,
Dave

**Chris
Hainsworth**

I was surprised and delighted to learn about being selected for the FIG UK Achievement Award for 2001. It is kind of you all to think that I have helped in some way.

We are really happy here in Spain and there are so many new things to do.

I wish you and everyone involved in FIG UK the very best wishes for the future.

Kind regards,

Chris Hainsworth

**Bill
Young**

Thanks very much for the email. I know that it is not much. I would like to do more but with work and family commitments, time just doesn't seem to stretch beyond these limits. However, Forth as a system/language/specification tool interests me greatly (actually, amazes me how simple yet powerful it is). I'm still trying to get Forth into a project at work, but

I enjoyed Graeme's articles on Finite State Machines and the Forth project board articles. Thanks.

For those that are interested, I have a copy of every Circuit Cellar (Steve Ciarcia's electronics magazine, www.circuitcellar.com) except the very first issue. I have found this magazine very helpful for hardware/firmware ideas/projects/information.

regards,

Bill Young

Sensors Scientist

Applied Technology Group

Sensors, Simulation and Automation Section

GlaxoSmithKline R & D



Chairman **Jeremy Fowell,** 11 Hitches Lane, EDGEBASTON B15 2LS
0121 440 1809 jeremy.fowell@btinternet.com

Secretary **Doug Neale,** 58 Woodland Way, MORDEN SM4 4DS
020 8542 2747 dneale@w58wmorden.demon.co.uk

Editor **Chris Jakeman,** 50 Grimshaw Road, PETERBOROUGH PE1 4ET
01733 352373 cjakeman@bigfoot.com

Treasurer **Neville Joseph,** Marlowe House, Hale Road, WENDOVER HP22 6NE
01296 62 3167 naj@najoseph.demon.co.uk

Webmaster **Jenny Brien,** Windy Hill, Drumkeen, BALLINAMALLARD,
Co. Fermanagh BT94 2HJ
02866 388 253 webmaster@figuk.plus.com

Librarian **Graeme Dunbar** Electrical Engineering, The Robert Gordon University,
Schoolhill, ABERDEEN AB10 1FR
01651 882207 g.r.a.dunbar@rgu.ac.uk

Membership enquiries, renewals and changes of address to Doug.
Technical enquiries and anything for publication to Chris.
Borrowing requests for books, magazines and proceedings to Graeme.

FIG UK Web Site

For indexes to Forthwrite, the FIG UK Library and much more, see <http://www.fig-uk.org>

FIG UK Membership

Payment entitles you to 6 issues of Forthwrite magazine and our membership services for that

period (about a year). Fees are:

National and international	£12
International served by airmail	£22
Corporate	£36 (3 copies of each issue)

Forthwrite Deliveries

Your membership number appears on your envelope label. Please quote it in correspondence to us. Look out for the message "SUBS NOW DUE" on your sixth and last issue and please complete the renewal form enclosed.

Overseas members can opt to pay the higher price for airmail delivery.

Copyright

Copyright of each individual article rests with its author. Publication implies permission for FIG UK to reproduce the material in a variety of forms and media including through the Internet.



FIG UK Services to Members

- Magazine** Forthwrite is our regular magazine, which has been in publication for over 100 issues. Most of the contributions come from our own members and Chris Jakeman, the Editor, is always ready to assist new authors wishing to share their experiences of the Forth world.
- Library** Our library provides a service unmatched by any other FIG chapter. Not only are all the major books available, but also conference proceedings, back-issues of Forthwrite and also of the magazine of International FIG, Forth Dimensions. The price of a loan is simply the cost of postage out and back.
- Web Site** Jenny Brien maintains our web site at <http://www.fig-uk.org>. She publishes details of FIG UK projects, a regularly-updated Forth News report, indexes to the Forthwrite magazine and the library as well as specialist contributions such as “Build Your Own Forth” and links to other sites. Don’t forget to check out the “FIG UK Hall of Fame”.
- IRC** Software for accessing Internet Relay Chat is free and easy to use. FIG UK members (and a few others too) get together on the #FIG UK channel every month. Check Forthwrite for details.
- Members** The members are our greatest asset. If you have a problem, don’t struggle in silence - someone will always be able to help. Do consider joining one of our joint projects. Undertaken by informal groups of members, these are very successful and an excellent way to gain both experience and good friends.
- Beyond the UK** FIG UK has links with International FIG, the German Forth-Gesellschaft and the Dutch Forth Users Group. Some of our members have multiple memberships and we report progress and special events. FIG UK has attracted a core of overseas members; please ask if you want an accelerated postal delivery for your Forthwrite.