$10 Volume XVII, Number 4 November 1995 December

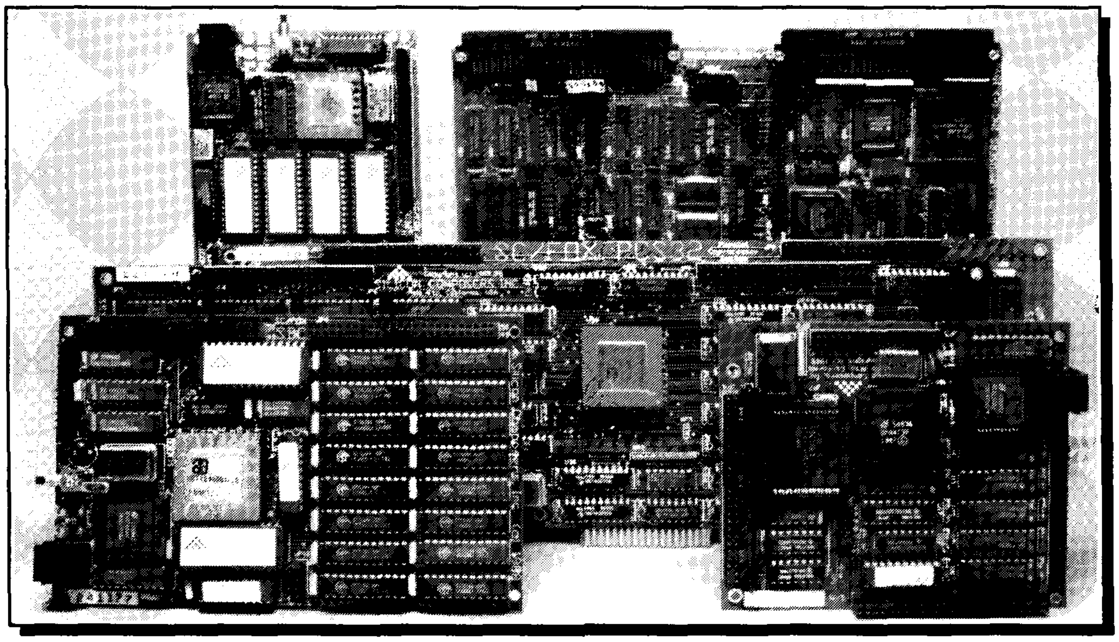
FORTH DIMENSIONS

SILICON COMPOSERS INC

FAST Forth Native-Language Embedded Computers

DUP

>R



C@

R>

Harris RTX 2000tm 16-bit Forth Chip
- •8 or 10 MHz operation and **15 MIPS** speed.
- •1-cycle 16 x 16 = 32-bit multiply.
- •1-cycle 14-prioritized interrupts.
- •two 256-word stack memories.
- •8-channel I/O bus & 3 timer/counters.

SC/FOX PCS (Parallel Coprocessor System)
- •RTX 2000 industrial PGA CPU; 8 & 10 MHz.
- •System speed options: 8 or 10 MHz.
- •32 KB to 1 MB 0-wait-state static RAM.
- •Full-length PC/XT/AT plug-in (6-layer) board.

SC/FOX VME SBC (Single Board Computer)
- •RTX 2000 industrial PGA CPU; 8, 10, 12 MHz.
- •Bus Master, System Controller, or Bus Slave.
- •Up to 640 KB 0-wait-state static RAM.
- •233mm x 160mm 6U size (6-layer) board.

SC/FOX CUB (Single Board Computer)
- •RTX 2000 PLCC or 2001A PLCC chip.
- •System speed options: 8, 10, or 12 MHz.
- •32 KB to 256 KB 0-wait-state SRAM.
- •100mm x 100mm size (4-layer) board.

SC32tm 32-bit Forth Microprocessor
- •8 or 10 MHz operation and **15 MIPS** speed.
- •1-clock cycle instruction execution.
- •Contiguous 16 GB data and 2 GB code space.
- •Stack depths limited only by available memory.
- •Bus request/bus grant lines with on-chip tristate.

SC/FOX SBC32 (Single Board Computer32)
- •32-bit SC32 industrial grade Forth PGA CPU.
- •System speed options: 8 or 10 MHz.
- •32 KB to 512 KB 0-wait-state static RAM.
- •100mm x 160mm Eurocard size (4-layer) board.

SC/FOX PCS32 (Parallel Coprocessor Sys)
- •32-bit SC32 industrial grade Forth PGA CPU.
- •System speed options: 8 or 10 MHz.
- •64 KB to 1 MB 0-wait-state static RAM.
- •Full-length PC/XT/AT plug-in (6-layer) board.

SC/FOX SBC (Single Board Computer)
- •RTX 2000 industrial grade PGA CPU.
- •System speed options: 8, 10, or 12 MHz.
- •32 KB to 512 KB 0-wait-state static RAM.
- •100mm x 160mm Eurocard size (4-layer) board.

For additional product information and OEM pricing, please contact us at:
SILICON COMPOSERS INC 655 W. Evelyn Ave. #7, Mountain View, CA 94041 (415) 961-8778

Contents

Editorial

The Programming Public

Over the years of my association with the Forth community, I've often heard the wish that more younger people would get involved. This, of course, means there is a desire to see a new generation to carry forward Forth practices and philosophy. In this issue, Nicholas Solntseff's synopsis of the Rochester Forth Conference includes the comment, "...registrants expressed hope that Open Boot Forth would lead to more neophytes at future meetings..."

Of course, for this to happen, the Forth community—including chapter leaders, conference organizers, writers, and others—must actually do something to capitalize on the opportunity. At FIG chapters, for instance, relevant presentations must be planned in advance, and the meetings must be publicized to capture the attention of those savvy to the implications of Open Boot/Firmware who are not already part of the Forth landscape. And follow-up activities must ensure that the public will find value in attending more than once. Other subjects, too, like embedded system development, target compilation, controlling remote hardware, and more, need to be presented and promoted in ways to attract the interest of the programming public in addition to serving our existing colleagues in Forth.

Maybe none of this sounds like as much fun as recompiling your kernel (again) but, without the major corporate or academic sponsorship enjoyed by other languages, Forth's growth depends—as it always has—on efforts at the grass roots. That means us, folks.

* * *

Some people mistakenly think of *Forth Dimensions* as printing the canonical truth according to the Forth Interest Group, but it ain't necessarily so, as some of my own editorials have demonstrated. *FD* is a vehicle for its readers to use, both to learn from and to contribute to; after all, FIG is member supported. Our community is diverse enough to engender the occasional disagreement on both technical and philosophical grounds, and it is mature enough also to be inclusive and try to find some value even in material to which we might individually take exception.

We are pleased to present opinion pieces in this issue by Peter Knaggs and Ronald T. Kneusel. We hope you will take them to heart—not as FIG gospel, but as food for serious thought. Let us know what you think, even submit your own opinion piece for publication. Sure, we usually prefer solid technical articles (and are always looking for more of them), but to misquote Tolkien, "All that glitters is not code." (Also, note that there was no collusion between Mike Elola and Ron Kneusel on their object-oriented articles published herein: this was just an interesting confluence of thinking.)

* * *

Everett (Skip) Carter has signed up to become our newest columnist. Skip is a professor at the U.S. Naval Postgraduate School, spearheads the Forth Scientific Library project, is a valued presenter at the FORML Conference, hosts the taygeta WWW and ftp Forth repository (site of FIG's home page), and generally has done a lot of solid Forth work. We are looking forward to presenting his contributions to you on a regular basis.

—*Marlin Ouverson*
FDeditor@aol.com

Forth Dimensions
Volume XVII, Number 4
November 1995 December

Published by the
Forth Interest Group

Editor
Marlin Ouverson

Circulation/Order Desk
Frank Hall

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

Subscription to *Forth Dimensions* is included with membership in the Forth Interest Group at $40 per year ($52 overseas air). For membership, change of address, and to submit items for publication, the address is: Forth Interest Group, P.O. Box 2154, Oakland, California 94621. Administrative offices: 510-89-FORTH. Fax: 510-535-1295. Advertising sales: 805-946-2272.

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

"*Forth Dimensions* (ISSN 0884-0822) is published bimonthly for $40/46/52 per year by the Forth Interest Group, 4800 Allendale Ave., Oakland, CA 94619. Second-class postage paid at Oakland, CA. POSTMASTER: Send address changes to *Forth Dimensions*, P.O. Box 2154, Oakland, CA 94621-0054."

Letters

Two Cents' Worth

Dear Marlin,

The July-August issue of *Forth Dimensions* waxed so philosophical that I may as well toss in my two cents' worth.

In the July issue of *Circuit Cellar, Ink* a short thread from their BBS began with a request by an embedded systems programmer who needed more information on Forth's ability to generate compiled code (as in C) versus working with a system using Forth in a microkernel. The Forthwrights answered favorably, but did not provide any source for more information.

Though the Forth Interest Group (FIG) aspires to be that resource, the items on the mail-order form are not enough. It should include a vendor list (perhaps extended to include ads) and a list of chapters and BBS's. This would provide a source for human contact. Better still would be setting up a FIG BBS and Web server. This can be supplemented by the Forth CD-ROM (vol. one), which would also provide a file resource for BBS's, as well as students and Forthwrights world wide.

The idea is to provide entry to Forth information at centralized locations which could more easily be promoted.

The new C++ compilers now sport something Forth has had for 25 years.

The new C++ compilers now sport Integrated Development and Debugging Environments (IDDE), something Forth has had for 25 years. Forth apparently reversed the usual compiler technology, beginning with the IDDE and ending with metacompilers to generate standalone programs.

In the July-August "Fast Forthward," Mike Elola ponders how to organize Forth. This brought to mind a text outliner from a decade past called Kamas, the brainchild of Adam Trent. This was also written in Kamas, a STOIC-like threaded interpretive language (TIL). It used a binary tree with labeled stems (indexes) to form the outline, and text blocks/pages for the leaves. The outline could expanded or contracted, and walked up or down at any level. The stems could be shifted to different levels, or tagged and then moved or copied. The tree/blocks were kept in files. If you could add the ability to use subfiles as stem extensions, you would have one cool block programming environment. Certainly better than dealing with a kazillion files.

Yours truly,
Walter J. Rottenkolber
P.O. Box 17005
Mariposa, California 95338

Walter:
Thank you for your letter! I recently re-instituted the list of Forth vendors with the help of L. Greg Lisle, and the list of on-line Forth resources (BBS or otherwise) with the assistance of Kenneth O'Heskin. I'm grateful for the generous (i.e., volunteer), diligent work of both those gentlemen, and hope the useful results move us in the direction you point out. Given the economics involved (limited number of pages and the desire to use them mostly for technical information), we may not publish the lists in every issue, but they will appear regularly and the FIG office now has them for use when answering inquiries.

—Editor

I don't question that a development environment such as the one you mention gives us the very nice hierarchical containers with which to organize code to the best of our abilities. I do question whether our code-organizing skills have been honed well enough that we are ready to exploit such an environment. I am reminded of how technology tends to arrive before political issues surrounding it are resolved (e.g., commercial use of the Internet).

Furthermore, I have been worried that satisfying load-order requirements may impose an upper limit on how well we can do. Donald Knuth developed the WEB code-writing environment to help address that and similar issues. See his book, Literate Programming.

I'd love to see you, or someone else who is aware of one, relate a "success story" that substantially tested the Kamas development environment.

—Mike Elola

From the Internet to Your Pocket:

Forth in the HP100LX

M. Edward Borasky
Beaverton, Oregon

What is the HP100LX?

The HP100LX Palmtop PC and its successor, the HP200LX, are DOS-compatible personal computers that fit in a jacket pocket and weigh 11 ounces. The CPU is an Intel 80186 running at 7.91 MHz. Memory can be either one or two megabytes (Mb). The user partitions this memory into two chunks: DOS/application memory and RAM disk. In the two Mb model that I have, the DOS memory ranges from 352 up to 636 Kb, leaving at worst 1.3 Mb of RAM disk which is accessed as the DOS "C:" drive. A ROM of two Mb in the HP100LX and three Mb in the HP200LX contains all the built-in software, some of which can be seen from DOS as the "D:" drive.

The screen, although monochrome, is CGA compatible and colors are mapped onto four grayscale values. The HP100LX displays text in three modes: 80 columns by 25 rows, 64 columns by 18 rows, and 40 columns by 16 rows. The 64x18 mode is the most practical compromise between convenience and readability, and works very well with Forth blocks of 16 rows of 64 columns. The standard PC mode of 80 columns by 25 rows is readable with good glasses and bright light, but is difficult to work with otherwise.

The HP100LX is the most powerful computer I own.

The HP100LX/200LX has one PC-compatible serial port, an HP-proprietary infrared port, and a single PCMCIA type II card slot. Available peripherals include printers, external floppy disk drives, numerous battery-backed SRAM and flash RAM PCMCIA cards for more RAM disk space, and FAX/data modems for either the PCMCIA or the serial port. My configuration at present consists of a two Mb HP100LX, a Sparcom Drive100 floppy disk drive which connects to the serial port, a Maxtor MobileMAX four Mb (nominal eight Mb with Stacker 3.0) PCMCIA flash memory card, and an EXP ThinFax 1414LX 14.4 Kbps FAX/data PCMCIA modem. The palmtop PC, its peripheral ports, and PCMCIA slot are powered by two AA alkaline or rechargeable NiCad batteries. I use an AC adapter/recharger for working with the modem but, for nearly everything else, an overnight charge cycle on NiCad batteries is good for about four-to-eight hours of continuous running. With intermittent use, a week on a charge is typical.

The built-in software includes a Phoenix-compatible BIOS, most of DOS 5.0, and 16 key-selectable applications. The major difference between the HP100LX and the HP200LX is the set of built-in applications supplied; for a Forth programmer, the two machines are essentially identical. The applications in both models include the Filer (very much like the File Manager in Windows), an appointment book, a phone book, a memo editor, an emulation of a Hewlett-Packard business calculator, a DOS prompt window for any DOS application (including Forth), and that timeless classic: Lotus 1-2-3 Release 2.4.

The built-in applications occupy separate segments of DOS memory when opened. The user can open as many as desired, as long as they all fit. Only a single application is active and on-screen at a time, but the user can switch among the open applications or open a new one with a single keystroke. Data can be cut or copied from one application and pasted into another using a clipboard. For applications that require the full 640 Kb DOS memory space, the user can terminate all the built-in applications and drop out to a pure DOS environment.

What do I do with it? First of all, the appointment book runs my life. Second, the modem is my main connection to the Internet away from work. Most of the software I tested for this article was downloaded this way. Third, the HP100LX is my primary vehicle for my various computational hobbies and technical writing. This article, for example, was typed on my HP100LX using the built-in memo editor. The benchmark tables you will see below were computed in Lotus 1-2-3, printed to a text file, and imported into the article with the memo editor. The HP100LX is the most powerful computer I currently own.

Before moving to my experiences with Forth in the HP100LX, a brief confession: not all of the programming and calculating that I do with my HP100LX is done in Forth (yet). I use the GNU "awk" processor ("gawk") heavily for extracting numerical data from ASCII files into convenient formats for Lotus 1-2-3, and for many simple calculations and preprocessing functions. I use the built-in Lotus 1-2-3 spreadsheet heavily for numerical and statistical processing, including the built-in multiple linear regression, and I use the Derive 3.0 math package for symbolic calculation and graphics. And, of course, I use the memo editor for creating and editing gawk scripts and other code files, including Forth text source files.

All of these tools do their jobs well, and if the HP100LX had a faster CPU and a larger address space, perhaps they would meet all my needs. But some of the things I do require more computational power than gawk or Lotus 1-2-3 are capable of delivering on a 7.91 MHz, 16-bit machine. Certain complicated calculations just take too long in gawk or Lotus 1-2-3. The HP100LX is, after all, a minicomputer rather than a mainframe. And the best, most efficient, and compact way to extract the optimum in performance and convenience from a minicomputer is the language that was created with the minicomputer's constraints in mind: Forth.

Forth in the HP100LX

Since the HP100LX is an 80186/DOS 5.0 environment, there are a number of public-domain, shareware, and commercial Forth packages available. Any 16-bit 8086/8088 DOS Forth will run in the HP100LX. I haven't tried them all, of course, but let me give some brief notes on those I have tested. All of the Forths I examined are either in the public domain or are low-cost shareware, and all but one were obtained using the modem over the Internet from various Forth FTP archives. In Appendix One, I've listed the Internet Uniform Resource Locator (URL) for each of the packages I tested from the Internet.

The availability of RAM disk space is often the main constraint in designing applications for the HP100LX. As noted above, I have 1.3 Mb on my "C:" drive, and another four to eight Mb on my flash memory "A:" drive. Accordingly, I have listed the Forths described below in order of increasing disk space requirements. In Table One, "zip size" is the size of the pkzip archive before unpacking. ".exe/.com size" is the type of executable (.exe or .com) and how much disk space it uses. "dir size" is how much space the directory occupies immediately after it is all unpacked. In many cases, unwanted files can be removed after unpacking, but the pkunzip process will abort if this much space is not available during the unpacking.

The smallest Forth I tested is Dr. Ting's eForth. The eForth executable is also the smallest, at 15600 bytes. eForth is very much a minimalist Forth. Like all of the interpreted Forths I tested, eForth is direct-threaded. There is no assembler or editor, and file input and output appear to be limited to redirected DOS standard input and output. eForth is designed to be simple and easily ported, even to such limited environments as microcontrollers. Both 8086 macroassembler and Forth source are provided, and both are well documented.

The version of eForth I tested came from the Forth Primer fprimer.zip. This package includes much tutorial material on Forth from various sources, plus two Forth interpreters: eForth and F-PC. eForth is in a file called eforth86.exe. This is a self-extracting archive; simply execute this program and it will create the rest of the files in the eForth package.

eForth seems like an ideal Forth for such environments as signal-processing chips, microcontrollers, and such, but it lacks most of the creature comforts a Forth application programmer expects. The intent of the eForth developers was to adhere to the ANSI standard wherever possible. However, some common core constructs, for example `DO ... LOOP`, are annoyingly absent. `DOES>` is also absent, although Forth code for it is included in the archive. eForth is a well-designed Forth kernel, and I recommend it highly for embedded systems and as an introduction to the process of Forth implementation.

In the quest for simplicity, a few decisions were made that will make eForth much slower than most other 8086 Forth implementations. For example, `UM*` (multiply two unsigned 16-bit numbers) was implemented with shifts and adds rather than with the 8086 MUL instruction. This is probably because eForth is associated with the P21/F21 chips, which do not have a multiply instruction. Most of the other 8086 Forths I tested use the MUL instruction. eForth is public domain. A disk and implementation manual are available from the author for a nominal charge of $25.

In programming in Forth on the HP100LX, my intent has been to use the ANSI standard whenever possible, which led me to the first Forth I used on the HP100LX, Martin Tracy's ZENForth. The version I use is 1.18B as distributed in Jack Woehr's book, *Forth: The New Model*[1]. ZENForth is a 16-bit, direct-threaded DOS Forth that is very close to the ANSI standard. Extensions and options in ANS Forth that make sense in the 16-bit DOS world are, for the most part, available. Two that aren't there that I would really like to see are an 8086 assembler for `CODE` words and a software floating-point package.

Both the traditional block files and modern ANS file I/O words are supported. Although I haven't used them yet, two features required for developing large programs—access to the full 80186 address space with long addresses and multiple word lists—are available in ZENForth. As a result, for jobs that

Table One. Forth sizes (bytes).

Forth	Archive Name	zip size	.exe/.com size		dir size
eForth	eforth86.exe	53999	.com:	15600	110132
ForthCMP	4cmp22s.zip	102600	.com:	32463	272105
ZENForth	zen18b.zip	76360	.exe:	66358	278489
hForth	hf86v092.zip	149718	.exe:	61488	540362
100Pygmy	100pygmy.zip	139722	.com:	16543	551094
F83	f83s6.zip	258541	.exe:	47379	1233157
TCOM	tcom25.zip	1279991	.exe:	205968	3118121
F-PC	fpc36.zip	1108172	.exe:	172144	3221166

can't be done with gawk or Lotus 1-2-3, ZENForth is my principal application development environment.

A typical ZENForth development session on the HP100LX works like this: I open a DOS window in 132 Kb and bring up ZENForth in it. Then I open the source file with the built-in memo editor and make whatever changes are needed. Switching from ZENForth to the memo editor is done with a single keystroke. After doing a SAVE in the memo editor with a single keystroke, another single keystroke switches me back to ZENForth. I type `INCLUDE` followed by the filename to compile the program, then the test sequence for the word I'm currently testing. If there are errors, ZENForth informs me; I then go back to the memo editor, fix the source, and do another SAVE.

My main debugging tools are `DUMP` and `.S`, both provided in ZENForth. ZENForth comes with complete source and documentation, and can be rebuilt with user modifications using the Borland Turbo assembler and linker. ZENForth supports the `SAVE` feature as well, so you can create new executable files with your own Forth code installed.

After I have an application running, if ZENForth isn't fast enough, I compile it directly to machine code with Tom Almy's ForthCMP. The archive is called 4cmp22s.zip. This is a target compiler which supports a compilable subset of ANS Forth. A Forth-83 version, 4cmp220.zip, is also available, but I haven't tested it. ForthCMP is shareware. Registration is $50, and this compiler is well worth it. It is, as we will see below, the fastest Forth I have found for the HP100LX.

In addition to the common Forth words, ForthCMP supports the ANSI facility and string word sets, long addresses, memory allocation and deallocation, four different memory models including ROM code, DOS files and other interface operations, and multitasking. The DOS file interface does not conform with the ANSI standard, but it's easy to write words to do the simple parameter adjustments.

Forth turns the HP100LX into a self-contained Forth development environment ... a pocket minicomputer.

There is also an 8086 assembler for defining `CODE` words, and several nice extensions for optimization of compiled Forth. As a result, applications written in Forth and compiled with the ForthCMP compiler are competitive in speed with those built using compilers for other languages.

How fast is it? As you will see from the benchmarks described below, ForthCMP ranges from three to 12 times as fast as ZENForth, with the average nearly five to one! The low end (three to one) is on code with many multiply and divide operations. The high end (twelve to one) is on code that is dominated by branching and looping overhead. With some hand tuning of the Forth code and use of some ForthCMP options, I expect to do better. This is an amazing Forth, even though it does not conform to the ANSI standard because it is not an interpreter. I recommend it highly for speed-sensitive applications.

Next in size is Dr. Wyong Koh's hForth. hForth is derived from eForth, but Dr. Koh has made a number of significant improvements. There are three memory models, "ROM," "RAM," and "EXE." The first two are compact and, like eForth, have very few primitives written in assembler. As a result, they are slower than most 8086 Forths. For the EXE model, however, Dr. Koh implemented nearly all of the core operations in assembler, and this version of hForth is competitive with other interpreted Forths. All three versions support more of the ANSI standard than eForth does, and an 8086 assembler is provided as an option. Like eForth, hForth is public domain. hForth shows that it is possible to maintain the simplicity of eForth without sacrificing performance or convenience.

Next, there is 100Pygmy, an adaptation of Pygmy for the HP100LX. Pygmy and 100Pygmy are shareware; the author, Frank C. Sergeant, asks $15 for registration. 100Pygmy is nearly identical to Pygmy. The only difference is that 100Pygmy was adapted to the 64x18 screen mode of the HP100LX by Robert S. Williams, MD. The 100Pygmy screen editor works very well on the HP100LX. Pygmy is another minimalist Forth; the current 1.4 release executable, including the Forth kernel, block/screen editor, metacompiler, and 8086 assembler, occupies only 16196 bytes.

Pygmy is based loosely on Charles Moore's cmForth. Code can be loaded either with the familiar block interface or with `INCLUDE` from text files. Pygmy comes with a wealth of tutorial information and on-line documentation, but the best way to learn Pygmy is to page through the 191 screens of the Forth source using the screen editor!

Forth is the most personalizable of languages, and Sergeant has made some choices that make porting and running existing Forth-83 or ANS Forth code difficult. For example, `DO ... LOOP` is absent from the base Pygmy, although Sergeant does provide code for this construct. `>R` is renamed `PUSH`, and `R>` is renamed `POP`.

In Pygmy, there are only two vocabularies, `COMPILER` and `FORTH`, and there are no `IMMEDIATE` words. This trait is inherited from cmForth. Normally, the colon compiler searches for a word in `COMPILER`, then `FORTH`. If the word is found in `COMPILER`, it is usually executed immediately; and if it is found in `FORTH`, it is compiled into the dictionary entry under construction.

The most unfortunate choice, though, is the use of the backslash (i.e., \) to force compilation of a word from the `COMPILER` vocabulary into a colon definition. While this is more compact than the traditional word `[COMPILE]` it replaces, it means that one cannot load any code that uses backslash to comment out the rightmost portion of a line! I use backslash-style comments heavily. As a consequence, I have not yet run any of my application code through Pygmy. This unfortunate non-compliance with recognized standards is the only flaw in an otherwise fine and compact Forth package.

Next in size is the venerable Laxen-Perry F83. The current version is public domain and the archive is called f83s6.zip. As the name implies, F83 conforms to the Forth-83 Standard. F83 has a metacompiler, an assembler, an editor, and a rich collection of I/O operations. Because of the superior development environment of F-PC, I have not put much effort into testing F83.

The largest Forth I have tested is version 3.6 of F-PC, fpc36.zip on the Internet. F-PC is primarily the work of Tom Zimmer, although a number of others have been involved and are credited in the source. Version 3.6 is the final version of F-PC; Zimmer and his colleagues have moved on to a 32-bit Forth for Windows systems. F-PC is public domain, with the exception that the floating-point packages are copyrighted by Dr. Robert Smith. Unlimited personal use is allowed, but embedding them in commercial software requires the consent of Smith. A disk version of F-PC is available from the author for $60.

F-PC is the most comprehensive Forth package in the public domain. At 3.2 Mb, it takes up most of my flash RAM disk. Although based on the Laxen-Perry F83, F-PC is a rich, complete, and comprehensive development environment, all built in Forth. Of course, there is a Forth-83 interpreter. There is also a combined editor/hypertext browser, a menu system including comprehensive on-line help, Forth and code word debuggers, a metacompiler, a decompiler and disassembler, an 8086 assembler, an ANS compatibility package, a word usage cross-referencer, a utility to search for strings in files, and a profiler for locating performance bottlenecks. As noted above, Robert Smith's two floating-point packages are included: an 8087-based hardware floating point and a software floating-point package using the traditional Forth 16-bit exponent and 32-bit mantissa. Complex arithmetic is also supported.

Once I backed up my "A:" and "C:" drives and removed all nonessential files, installation of F-PC from a floppy disk was simple. It was also easy to configure F-PC for the monochrome-mapped CGA screen of the HP100LX. F-PC is screen oriented and works well in the 80x25 mode on the HP100LX, provided the lighting is bright enough.

Normally, I terminate all the built-in HP100LX applications and run F-PC from a DOS prompt; some of the hypertext files will not fit in memory when running F-PC along with built-in applications. The F-PC editor and other built-in tools are very good and eliminate the need for external tools in the development cycle. The on-line help and source files are very impressive, including pictorial representations of the dictionary structure! F-PC is a magnificent example of the power of Forth.

Finally, I tested Tom Zimmer's F-PC target compiler, TCOM. TCOM is public domain and is currently available in release 2.5. On the Internet, the archive is tcom25.zip. Like ForthCMP, TCOM translates Forth source directly into executable 8086 code. Most of the things that work in F-PC also work in TCOM. TCOM only generates .com executables, which are limited to a single 64 Kb segment. However, TCOM programs can access the whole DOS memory space at run time, just like F-PC programs. I do not have enough RAM disk space to keep the full F-PC and TCOM resident at the same time, so I did not do any extended testing.

TCOM comes with a source/assembly-level debugger, a profiler, extensive documentation, and numerous sample programs. There is also a software floating-point package. The floating-point package comes from Dr. Robert Smith, who wrote the F-PC packages. The two appear to be different, however. The one in TCOM emulates IEEE arithmetic and is probably much slower as a result.

Benchmarks

Once I had some idea of the size and feature range of low-cost Forths for the HP100LX, I decided to go further and try to assess the speed of them as well. Tom Almy's ForthCMP comes with two Forth benchmarks, one from *Byte* magazine and the other from *Interface Age*, so I decided to run these with all the Forths. Both make heavy use of `DO ... LOOP`, and the *Interface Age* benchmark also uses `LEAVE`. Thus, neither will run with eForth, and only the *Byte* magazine benchmark will run with 100Pygmy.

In order to get a second speed assessment for 100Pygmy and to get a measure of absolute performance, I wrote a third benchmark, using `BEGIN ... UNTIL` loops, which all the Forths support. My benchmark is a small Forth vector arithmetic test inspired by the Livermore Fortran Kernels [2], a well-known supercomputer benchmark. There are five loops: vector no-operation, vector add ("+"), vector multiply ("*"), vector divide ("/"), and vector scalar multiply ("*/"). There are 1000 repetitions of each operation, and each operation processes vectors of length 1000, so a million operations are performed in each of the five doubly-nested loops.

The combined source file for the ZENForth/ANS version of all three benchmarks is shown in Appendix Two. The *Byte* magazine benchmark as distributed with ForthCMP has an unfortunate defect. It stores and reads flags in an area of memory referred to by absolute decimal address 10000! While this may work with ForthCMP, it may not with other systems and I have replaced it with an `ALLOT`ed array.

Table Two shows the benchmark timings from the HP100LX.

Table Two. Benchmark run times (seconds — 7.91 MHz 80186).

	ForthCMP	TCOM	hForth	Pygmy	ZEN	F-PC	F83
Byte magazine	31	107.88	225	247	234	281.77	421
Interface Age	35	85.79	156	N/A	187	205.14	316
Vector No-Op	3	3.63	36	30	36	46.41	68
Vector +	81	99.03	210	255	264	348.50	402
Vector *	85	104.96	215	250	260	354.05	421
Vector /	89	118.42	253	263	333	363.44	814
Vector */	72	102.49	246	205	279	325.71	911

Table Three. Relative times (ForthCMP = 1.0).

	ForthCMP	TCOM	hForth	Pygmy	ZEN	F-PC	F83
Byte magazine	1.0	3.5	7.3	8.0	7.5	9.1	13.6
Interface Age	1.0	2.5	4.5	N/A	5.3	5.9	9.0
Vector No-Op	1.0	1.2	12.0	10.0	12.0	15.5	22.7
Vector +	1.0	1.2	2.6	3.1	3.3	4.3	5.0
Vector *	1.0	1.2	2.5	2.9	3.1	4.2	5.0
Vector /	1.0	1.3	2.8	3.0	3.7	4.1	9.1
Vector */	1.0	1.4	3.4	2.8	3.9	4.5	12.7
Geometric Mean	1.0	1.6	4.2	4.3	4.9	6.0	9.7

Appendix One. URLs of Forths tested.

ForthCMP:	ftp://oak.oakland.edu/simtel/msdos/forth/4cmp22s.zip
F83:	ftp://ftp.cygnus.com/pub/forth/f83s6.zip
eForth:	ftp://ftp.cygnus.com/pub/forth/fprimer.zip (contains eForth86)
F-PC:	ftp://taygeta.com/pub/Forth/Reviewed/fpc36.zip
hForth:	ftp://taygeta.com/pub/incoming/forth/hf86v092.zip
TCOM:	ftp://taygeta.com/pub/Forth/Reviewed/tcom25.zip
100Pygmy:	ftp://eddie.mit.edu/distrib/hp95lx/hp100lx/100pygmy.zip

Having established that Tom Almy's target compiler ForthCMP produces the fastest run times, I computed the ratios of the benchmark times for each of the other Forths to the time for ForthCMP. Then I computed the average of these ratios for each Forth to estimate relative performance. The correct average to use when averaging relative benchmark timings like these is the *geometric mean*, not the common average or arithmetic mean [3].

Suppose you have N relative times T(1), T(2), ... T(N). The geometric mean is simply the exponential of the arithmetic mean of the natural logs of the times! Alternatively, it is the N-th root of the product of the relative times, but the logarithmic method is much easier to do in a spreadsheet. Table Three gives the computed relative times and their geometric means.

Summary

As we have seen, a wide range of low-cost Forth packages are available for the HP100LX, all of which turn the HP100LX into a self-contained Forth development environment. Moreover, with target compilation, it is easy to develop high-speed programs without using CODE words. In short, Forth turns the HP100LX into a pocket minicomputer. I sometimes refer to the HP100LX as the Volkswagen Beetle of the Information Superhighway.

Unfortunately, space does not permit me to go into much detail on my applications, which mostly deal with the mathematical analysis of stock and futures trading systems. The largest project I've completed so far is my entry in the *NeuroVe$t Journal* International Nonlinear Financial Forecasting Contest. This program used an ancient technical analysis method known as *point and figure charting*. I have not seen the final results yet, but I expect my code in ZENForth on the HP100LX to compete with more complex neural net algorithms running on a 33 MHz '486.

References

[1] Woehr, Jack (1992), *Forth: The New Model*, M & T Books, San Mateo, California, ISBN 1-55851-277-2.

[2] McMahon, Frank H. (1986) *The Livermore Fortran Kernels*, Lawrence Livermore National Laboratory, Livermore, California, UCRL-53745, December 1986.

[3] Fleming, Philip J., Wallace, John J. (1986), "How Not To Lie With Statistics: The Correct Way to Summarize Benchmark Results", *Communications of the ACM*, Volume 29, Number 3, pages 218–221.

M. Edward Borasky is an applied mathematician and computer scientist who has written software for machines ranging from programmable calculators to massively parallel supercomputers. His interests include computer music, computational finance, computer system performance analysis and, of course, the Forth language. He currently works for a major vendor of turnkey business computers as a UNIX performance guru. Although his desk has been declared a Forth-free zone, his palmtop PC has been granted an exemption. He can be reached at znmeb@teleport.com or at http://www.teleport.com/~znmeb (his home Web page).

ADVERTISERS INDEX

FORML Conference	44
FORTH, Inc.	23
Forth Interest Group	centerfold
Laboratory Microsystems, Inc. (LMI)	21
Miller Microcomputer Services	21
Offete Enterprises	12
Silicon Composers	2

Appendix Two. ZENForth/ANS benchmark source.

```
( system-dependent timing routines )
( this is the ForthCMP/ZEN/ANS version )
( output doubles are seconds since midnight )
: GETTIME ( -- sec min hour )
  TIME&DATE ( -- sec min hour day month year )
  2DROP DROP ( ditch date part )
;

: T>B ( sec min hour -- d )
  60 * + ( sec min )
  60 UM* ROT S>D D+
;

: BENCH ( start timing )
  ( -- d )
  CR GETTIME T>B 2DUP D. ( look at clock )
;

: MARQUE ( stop timing )
  ( d -- )
  GETTIME T>B 2DUP D. D- DNEGATE D.

( BYTE MAGAZINE BENCHMARK )
;

8190 CONSTANT SIZE
VARIABLE BFLAGS SIZE ALLOT

( defined as intrinsic in "ForthCMP" )
: C<- SWAP C! ;

: DO-PRIME
  BFLAGS SIZE 1 FILL
  0 SIZE 0 DO
    BFLAGS I + C@ IF
      I 2* 3 + DUP I + BFLAGS +
      BEGIN
        DUP SIZE BFLAGS +
      U< WHILE
        DUP 0 C<- OVER +
      REPEAT
      DROP DROP 1+
    THEN
  LOOP
  U. ." PRIMES" CR
;

: BYTEBNCH
  ." 100 ITERATIONS"  CR
  BENCH 100 0 DO DO-PRIME LOOP MARQUE
;

( INTERFACE AGE BENCHMARK 08:01 11/16/85 )
( This is the Interface Age benchmark )
( program described in Appendix D of the )
( ForthCMP Manual. )

: IABENCH DUP 2/ 1+ SWAP CR
  1 DO
    DUP I 1 ROT 2 DO
      DROP DUP 0 I UM/MOD DUP 0= IF
        DROP DROP 1 LEAVE
      THEN
      1 = IF
        DROP 1
      ELSE
        DUP 0= IF
          DROP 0 LEAVE
        THEN
        0< 0= IF
          1
        THEN
      THEN
    LOOP
    IF
      .
    ELSE
      DROP
    THEN
  LOOP
  DROP CR
;

: IAGEBNCH
  BENCH 5000 IABENCH MARQUE
;

( VECTOR LOOP BENCHMARK )
( M. EDWARD BORASKY )
( 30 JULY 1995 )

( uses BEGIN … UNTIL loops; all tested )
( Forths have them )
( some small Forths are missing DO … LOOP )
( or FOR … NEXT )

1000 CONSTANT VSIZE ( vector size )

: CELLS 2* ;

: VECTOR ( make an array )
  ( n -- ) ( compiling -- reserve memory )
  CREATE CELLS ALLOT

  ( index -- address )
  ( executing -- compute address )
  DOES> SWAP CELLS +
;

VSIZE VECTOR VEC1 ( vector 1 )
VSIZE VECTOR VEC2 ( vector 2 )
VSIZE VECTOR VEC3 ( vector 3 )

: VECLOAD ( put some stuff into the vectors )
  ( -- )
  0 BEGIN
    DUP VEC1 DUP !
    ( VEC1 gets its own address )
    DUP VEC2 DUP NEGATE SWAP !
    ( VEC2 gets NEGATEd address )
    1+ DUP VSIZE =
  UNTIL
  DROP
;

: LOOP0 ( Null loop )
  ( -- )
  0 BEGIN
    1+ DUP VSIZE =
  UNTIL
  DROP
;

: LOOP1 ( Vector Add )
  ( -- )
  0 BEGIN
    DUP VEC1 @ OVER VEC2 @ + OVER VEC3 !
    1+ DUP VSIZE =
```

(Code continues on next page.)

```
  UNTIL
  DROP
;

: LOOP2 ( Vector Multiply )
  ( -- )
  0 BEGIN
    DUP VEC1 @ OVER VEC2 @ * OVER VEC3 !
    1+ DUP VSIZE =
  UNTIL
  DROP
;

: LOOP3 ( Vector Divide )
  ( -- )
  0 BEGIN
    DUP VEC1 @ OVER VEC2 @ / OVER VEC3 !
    1+ DUP VSIZE =
  UNTIL
  DROP
;

: LOOP4 ( Vector Scale )
  ( -- )
  0 BEGIN
    DUP VEC1 @ 10000 10000 */ OVER VEC2 !
    1+ DUP VSIZE =
  UNTIL
  DROP
;

1000 CONSTANT REPS ( repetitions )

: BENCH0 ( benchmark LOOP0 )
  ( -- )
```

```
  BENCH
  0 BEGIN
    LOOP0
    1+ DUP REPS =
  UNTIL
  DROP
  MARQUE
;

: BENCH1 ( benchmark LOOP1 )
  ( -- )
  BENCH
  0 BEGIN
    LOOP1
    1+ DUP REPS =
  UNTIL
  DROP
  MARQUE
;

: BENCH2 ( benchmark LOOP2 )
  ( -- )
  BENCH
  0 BEGIN
    LOOP2
    1+ DUP REPS =
  UNTIL
  DROP
  MARQUE
;

: BENCH3 ( benchmark LOOP3 )
  ( -- )
  BENCH
  0 BEGIN
    LOOP3
    1+ DUP REPS =
  UNTIL
  DROP
  MARQUE
;

: BENCH4 ( benchmark LOOP4 )
  ( -- )
  BENCH
  0 BEGIN
    LOOP4
    1+ DUP REPS =
  UNTIL
  DROP
  MARQUE
;

: LOOPBNCH
  VECLOAD
  BENCH0
  BENCH1
  BENCH2
  BENCH3
  BENCH4
;

: MAIN
  BYTEBNCH IAGEBNCH LOOPBNCH
;

( run benchmarks and exit back to DOS )
MAIN
BYE
```

Hashing Forth

Xan Gregg
Durham, North Carolina

Hashing provides a fast way to search a large, unsorted data set at the cost of extra memory. Robert Sedgewick, in his book *Algorithms*, concisely describes hashing as "directly referencing records in a table by doing arithmetic transformations on keys into table addresses." That should make sense to you by the end of this article, but first, let's consider a simple example.

Suppose you have to write code to manage a database of about 50,000 records referenced by 16-bit record numbers. Record insertions and deletions are common, so they can't be too slow, and record look-ups are frequent and must be fast. You are given 64K of RAM in addition to the memory and disk space occupied by the data, and you know that each record is referenced by a unique three-letter code, like airports are in the U.S.

As a Forth programmer, you realize that a three-letter string is also a three-digit base-26 number, and you make a table with 26 x 26 x 26 = 17,576 entries, with each entry containing the record number. Insertion and deletion are straightforward—you just have to update the table with each operation. Finding a record from its key involves only packing three letters into a 15-bit number and using it as an index into the table. Then you have the record number.

The time/space tradeoff is what hashing is all about.

If you can do that, you already understand the basic concepts of hashing. Hashing requires a hash function and a hash table. The hash function converts a key value, such as a text string or a large number, into a hash table address. Each entry in the hash table points to a record in the data set.

In the example above, the code to pack three letters into a 15-bit number was the hash function, and the table of record numbers was the hash table. The hash function might look like Figure One.

However, since you have 64K of memory and look-up speed is so important, you could multiply by 32 instead of 26 so that you could use a shift operation by changing each `26 *` into `5 LSHIFT`. Having an efficient hash function is very important, and it is often written in assembler.

Hashing becomes more interesting when the key is too big to be packed into a number and when the keys are not unique. For example, what if the key was a person's last name? That's more realistic than the unique, three-letter key given above. It's not obvious how to use hashing in this situation. We need a hash function to convert a variable-length string into a table index, and we need a way of dealing with multiple records that are mapped to the same table index.

The hash function can be almost anything. Here are some possibilities for the last-name key mapping into a 15-bit number:

1. Use first three letters, five bits per letter.
2. Use last three letters, five bits per letter.
3. Use first five letters, three bits per letter.
4. Multiply all letters, modulo 2^{15}.
5. Use five bits from first letter, two bits from next five letters.
6. Use three bits from length byte, three bits from first four letters.
7. Sum all letters, modulo 2^{15}.
8. Sum all letters $letter_i * 2^i$, modulo 2^{15}.

I'm sure you can imagine more. The hard part is picking a good one. You want a function that is quick to execute and provides a fairly random distribution of keys. #1,

Figure One. Example hash function.

```
: ID>INDEX  ( addr -- n ) \ addr points to three uppercase letters
   COUNT [CHAR] A -          \ addr+1 n1
   SWAP COUNT [CHAR] A -     \ n1 addr+2 n2
   SWAP C@ [CHAR] A -        \ n1 n2 n3
   26 * + 26 * + + ;         \ n
```

Table One. Average list lengths of various hash functions.

Hash Function	W/1024	W/4096
Low bits of first three chars	5.23	3.44
Low bits of length and first two chars	4.36	2.30
Low bits of last three chars	6.31	4.41
Product of first three chars	6.64	3.91
Sum of $char_i * 2^i$	3.56	1.57

above, is not good because many of the names probably start with the same letters, providing a poor distribution. #4 is bad for two reasons: multiplying is usually not a good idea for speed's sake, and the distribution is not good because multiplying several numbers rarely produces odd numbers. #7 is bad because the sum of all of the letters will not approach 215. #8 tries to overcome this shortcoming by incorporating a shift into the add, and it may do well with some fine-tuning.

Any of the other hash functions may also do well with tuning. For instance, when you take three bits from a letter, how you do decide which three bits? The high three bits is obviously no good, since all upper-case letters start with the same three bits. The low three bits seems okay, but it's not perfect. H, P, and X map to 0; B, J, R, and Z map to 1; and G, O, and W map to 7; and those hardly seem like equal groupings.* The only way to know for sure is to try each possibility on the entire data set and see which one produces the best distribution. Unfortunately, you often have to develop your code with only a subset of the data, which requires you to rely more on your intuition.

Before getting too deep into selection of hashing functions, we need to address the second problem: what to do when two or more keys share the same index into the hash table, which is called a *collision*. There are several solutions, but the one I prefer is to add a field to each record that can point to another record. Then each table entry becomes the head of a linked list, with the other links being in the new field we added to each record.

I implemented hashing in MacForth and PowerMacForth, and several other Forths also use hashing.

Another solution is to use a secondary hash function to produce another index or just add a number (relatively prime to the table size) to the index to get a new index. In either case, you have the possibility of another collision, which requires finding another hash index, and so on. This solution has the requirement that the hash table have at least one entry for each record.

Either case requires more work during the search. You must compare the key against the record found, and, if it is not equal, you must check the next record, whether from the linked list or from another hash table entry. And similarly, insertion and deletion become more complicated as well.

So hashing isn't quite as simple as it appeared in our first example, but it is still very useful when search time is critical. A classic usage for hashing is in compiler symbol tables. A program may have thousands of symbols which a compiler must look up very frequently by name as it scans the source code. I implemented hashing in the MacForth and PowerMacForth vocabularies, and I know several other Forths also use hashing.

Listing One contains some generic hashing code. `FILL-TABLE` inefficiently allocates records on the fly in the data space, but the basic words (`INSERT-RECORD`, `DELETE-RECORD`, and `FIND-RECORD`) provide a good initial base for anyone trying to get started with hashing. The code uses the linked-list approach for collision handling.

`ANALYZE-HASH` is a useful word for getting some information about how well the hash function maps the keys. It produces lines numbered zero through nine which show how many linked lists there are of that size (or more, in the case of nine), and another line that shows the average list length (not counting the empty lists). You want to minimize that value. Another useful analysis tool I have used in the past is to plot the hash table showing the list length at every entry. Sometimes that reveals patterns of indices that are never hit or that are hit too often.

Table One gives the average list lengths for various hash functions given a data set of 3448 Forth words and hash table sizes of 1024 and 4096 entries.

All of these hash functions provide decent results. A binary search of the same data would require us to look at log2 3448 = 11.8 records. The last one is very close to optimal (3448/1024 = 3.37), and is a favorite of mine. It is similar to the function used in PowerMacForth, and it has the advantage that many characters factor into the result.

This table is the first mention of hash table size since our original, 64K hash table. The more entries you have in the table, the better your hashing will work (because there will be fewer collisions), but, of course, it takes more memory. The time/space tradeoff is what hashing is all about.

Happy hashing!

*But who knows? Maybe there are as many Greggs, Ouversons, and Wests as there are Browns, Joneses, Rathers, and Zwickkers.

Readers can contact Xan Gregg at his xgregg@aol.com e-mail address. He is a Macintosh programmer at Scholastic, Inc. working on educational software and living in Durham, North Carolina. Xan has been programming with MacForth since 1984 and, in his free time, he plays Ultimate Frisbee.

Listing One. Generic hashing code, with analysis words.

```
anew --hash--

( Generic Hashing Code
  Each record must start with one cell for use by the hashing code.
  It must be followed by a counted string which is the key.
  Variable-length data may optionally follow. )


: ALLOTERASE   ( n -- )   \ utility word
      HERE SWAP DUP ALLOT ERASE ;

1024 CONSTANT /HASHTABLE
CREATE HASHTABLE
      /HASHTABLE CELLS ALLOT


: INIT-HASHING   ( -- )
      HASHTABLE /HASHTABLE CELLS ERASE ;

: MY-HASH-FUNCTION   ( key-addr -- table-index )
      \ you should override this function
      COUNT 0 SWAP 0 DO            \ addr cur-index
            SWAP COUNT I LSHIFT ROT +
      LOOP NIP
      /HASHTABLE 1- AND ;          \ assumes power of 2

: KEY>INDEX   ( key-addr -- table-index )
      MY-HASH-FUNCTION
      0 MAX /HASHTABLE 1- MIN ;    \ for safety during development

: INSERT-LINK   ( recAddr tableAddr -- )
      \ insert at top of linked list for this index
      DUP @ ROT DUP >R !           \ recAddr.link <= previous top
      R> SWAP ! ;                  \ top <= recAddr

: INSERT-RECORD   ( recAddr -- )
      DUP CELL+ KEY>INDEX CELLS HASHTABLE + INSERT-LINK ;

: DELETE-LINK   ( recAddr tableAddr -- )
      \ remove link from list
      SWAP >R
      BEGIN
            DUP @ R@ <> SWAP @ 0 <> AND
      WHILE
            @     \ no match, so go to next link
      REPEAT
      DUP @ 0 <> IF
            R> @ SWAP !            \ prev.link <= rec.link
      ELSE
            R> DROP                \ do nothing if not found
      THEN ;

: DELETE-RECORD   ( recAddr -- )
      DUP CELL+ KEY>INDEX CELLS HASHTABLE + DELETE-LINK ;
```

(Code continues on next page.)

```
: FIND-LINK   ( addr tableAddr -- recAddr or 0 )
      >R BEGIN
            @ DUP
      WHILE
            DUP 1 CELLS + COUNT
            R@ COUNT COMPARE 0= IF
                  R> EXIT           \ found match, so exit
            THEN
            @                       \ no match, so go to next link
      REPEAT ;

: FIND-RECORD  ( addr -- recAddr or 0 )
      DUP KEY>INDEX CELLS HASHTABLE + FIND-LINK ;

\ ---- analysis words ----
: FILL-TABLE ( -- )
      INIT-HASHING
      S" FILEDATA1" R/O OPEN-FILE IF ABORT THEN
      >R    \ stash the file-id
      BEGIN
            HERE 32 CELL+ ALLOTERASE      \ space for text and link
            DUP CELL+ 1+ 31 R@ READ-LINE
            0= OVER 0 <> AND
      WHILE
            DROP                          \ recAddr bytesRead
            DUP IF
                  OVER CELL+ C!
                  INSERT-RECORD
            ELSE
                  2DROP
            THEN
      REPEAT 2DROP DROP
      R> CLOSE-FILE DROP ;

10 CONSTANT MAX-DEPTH
CREATE DEPTHS MAX-DEPTH CELLS ALLOT
VARIABLE TOTAL-ENTRIES
VARIABLE TOTAL-LISTS

: COUNT-LINKS  ( tableAddr -- n )
      0 SWAP BEGIN @ DUP WHILE SWAP 1+ SWAP REPEAT DROP ;

: ANALYZE-HASH  ( -- )
      MAX-DEPTH 0 DO I CELLS DEPTHS + OFF LOOP
      TOTAL-ENTRIES OFF
      TOTAL-LISTS OFF
      /HASHTABLE 0 DO
            I CELLS HASHTABLE + COUNT-LINKS
            DUP TOTAL-ENTRIES +!
            DUP 0> IF 1 TOTAL-LISTS +! THEN
            MAX-DEPTH 1- MIN
            1 SWAP CELLS DEPTHS + +!
      LOOP
      MAX-DEPTH 0 DO
            CR I 3 .R 2 SPACES I CELLS DEPTHS + @ 5 .R
      LOOP CR ." AVE = "
      TOTAL-ENTRIES @ 100 TOTAL-LISTS @ */
      S>D <# # # [CHAR] . HOLD #S #> TYPE ;
```

OOP, Forth, & the Future

Ronald T. Kneusel
Milwaukee, Wisconsin

Heraclitus would have loved Forth, it is the one computer language that is always changing. Yet, unless it changes rapidly and dramatically, Forth is in serious danger of missing the opportunity of its lifetime. Most of the computing industry threw its hat in the C ring long ago and now seems to be moving to C++. Why? Whatever its drawbacks, C++ is deemed more powerful than C because of its object-oriented features, and users are demanding ever-more-powerful applications. The object-oriented (OO) paradigm is winning the day, and C++ is the clear forerunner in the fight. But need it be? Many who use C++ do not appear to like it. This is key... an Achilles heel that, in my view, opens the door for a new era for Forth.

The world has gone GUI. It is a fact of life, an undeniable reality. True, embedded systems are pretty much the same as they have always been, but I'm thinking here of expanding Forth's horizons into areas that have not been traditional Forth territory. GUI operating systems beg for object-oriented languages to interface with, and Forth is perfectly suited to this new world.

The Forth community must move quickly or lose, perhaps forever.

For the most part, I use the Macintosh. Presently, there are two freeware, object-oriented Forths available for the Macintosh: Yerk and Mops. Offspring of the commercial language Neon, both interface well with the Macintosh operating system, providing a nice way to get things done. Yerk is presently used at Yerkes Observatory for fantastic things involving everything from remote telescope control, to TCP/IP, and imaging. Forth need not be stuck in a decades-old model, as is C. The marriage of Forth and object-oriented techniques provides considerable power. One programmer with an object-oriented Forth could do things that might take an entire team of C programmers much longer to accomplish, and have a final result that would be easier for a future programmer to understand and maintain.

Think about some of the reasons for an object-oriented Forth at the present time:

- Hit your enemy when he's confused. During this period of transition from C to C++, Forth has an opportunity that may not come again.
- New operating systems are emerging that do not fit well with the current crop of programming languages. Some companies, like Apple Computer, are inventing new ones (e.g., Dylan), but an object-oriented Forth would work as well.
- New hardware architectures are emerging that will use Forth. Operating systems and applications based on an object-oriented Forth would have a definite advantage.

Now think about a few of the possibilities an object-oriented Forth could create:

- An object-oriented operating system using, and written in, Forth. In a sense, this is a step into the past, but with a new twist that dramatically increases the possibilities. One reason for the success of Unix was its close ties with C; the same would hold true here.
- Forth is always playing the game of "See, I can do it, too..." like a little brother copying his elder sibling and demanding that everyone in the family pay attention to that fact. Yes, it is fun to write a particular application in Forth just to see it done in Forth, but the world will not be claimed by those who merely copy what has already been done, even if it is faster and smaller. An object-oriented Forth would soon have the other guys trying to say, "See, C++ can do it, too."
- An object-oriented Forth could be successfully introduced into the universities. More Forth courses taught = more Forth programmers.
- In scientific applications, an object-oriented Forth (with a nice infix expression compiler!) could start to displace Fortran, especially as scientists experience the power of a truly interactive system. Adding a simple interface to the mountains of existing Fortran code, and a sharp target compiler for finished applications, would help immensely. This area is largely untapped by industry. Standard Forth will work here, but the addition of objects would make it that much easier to get researchers to want to learn Forth.

As I see it, Forth is in some ways stuck in the early 1980s. Forth has a choice: to tag along with C and die a slow COBOL-esque death, or to jump ahead and take the front line. Sometimes the old becomes new again (look at neural networks, for example) and it seems the industry is just waiting for something like that: somewhat familiar, yet fresh and new and powerful. Marrying OOP and Forth is a logical thing to do.

A standard is needed that includes, at a minimum, an object-oriented wordset for Forth. Perhaps it might be time to take the Forth philosophy and move it to an entirely "new" language with its own standard, apart from what Forth currently is. Either way, the Forth community must move quickly or lose, perhaps forever. It must be an organized move by the entire community if it is to succeed. Speed, power, flexibility, modularity, and reusability are nice, but if there is no widespread standard, there will be no widespread success. People will avoid it for fear of being stuck with an oddball system.

How does one infiltrate and conquer the world of C? This is an involved question, but I see two immediate necessities: (1) a seamless interface between C libraries and Forth, and (2) a C to object-oriented Forth compiler. The former is definitely easier than the latter, but the compiler, if available, might help convince those with an existing C product to move it to the object-oriented Forth and not to C++. The world was once Fortran/COBOL and moved to C; it can make the adjustment, if given sufficient reason for doing so. A well-planned and executed object-oriented Forth might provide this reason.

With an object-oriented Forth, nothing is lost and much is gained in terms of flexibility, productivity, and ease of development. Moreover, much is gained in buying a future for Forth itself. It is time for Forth to take the lead, and adopting the object-oriented paradigm will allow it to do so.

Ronald T. Kneusel works as a systems specialist for the Division of General Internal Medicine at the Medical College of Wisconsin, Milwaukee, where he also spends about 50% of his time doing data manipulation and analysis for a research group. He has used Forth to write several applications for the Macintosh, all of which are available via the Web from http://141.106.68.98/ or via FTP at 141.106.68.98. He can be contacted by mail at 8725 West Burdick Avenue, Milwaukee, Wisconsin 53227 U.S.A., and by e-mail at rkneusel@post.its.mcw.edu.

Welcome to the New Forth Interest Group Board of Directors

I want to thank all the ten nominees for the Board of Directors for standing for election and making their interests in FIG known. This was the first election that had gone to a ballot of the members, and though the choices weren't easy to make, the members have decided.

I want to congratulate the nine who were elected. Elected were:

To the Board:

The job of the Board of Directors is not at all an honorary one. The board has always been a hard-working board, and this board has its work cut out for it.

FIG is now in its 18th year.

We are publishing Volume 17 of *Forth Dimensions.* That is 17 years of on-time, bi-monthly Forth papers, articles and news, edited and published by FIG with the material provided by Forth users and researchers and paid for by FIG members. This has been a magazine as good as any found on the newsstands, and is unequaled as a user group journal.

This November we will be holding our 17th annual FORML conference. Seventeen years hosted by FIG where attendees have provided over 640 major papers for the Forth community. Annual FORML proceedings are published by FIG. FORML is where those who have the fortune to attend are able to "hobnob" with their fellow wizards. FORML makes the prior year's routine work worthwhile.

Chapters have been the place in lieu of FORML for the rest of the year and for those that cannot attend, where the real face-to-face action of Forth takes place. New and old ideas can be presented, and new and seasoned members can come and learn from each other. Loyal support by chapter members has made FIG and will decide the ultimate fate of FIG.

Worldwide membership of loyal Forth people has molded FIG into what it is. Without their support, there would be nothing and the world would be poorer for not having Forth. We have a strong, loyal membership and if we provide the leadership, they will provide the will and the strength.

Because of the need to maintain an *FD* subscription base, support FORML registration, provide distribution of Forth literature, and maintain a point of contact for Forth users, FIG has maintained an office and office personnel. The office's job is to contact and remind the members of dues, provide available Forth literature, support members' needs and problems, listen and re-route the technical questions, and provide financial reporting.

All of the above has required direction and coordination to exist. Some set on their course have needed less attention, others have needed more, and yet others have needed more than we have been able to give and have withered. The new Board's challenge is to continue, to change, and to redress at the same time.

The Board of Directors will hold its first face-to-face meeting after FORML in November. Some of the first things the Board needs to do is set a course for the next few years for FIG and choose officers who can steer that course.

To the Members:

Please provide the Board and its officers your support, ideas, shoulder, and encouragement. We will need it in our 19th and future years. Input to the new board members will help determine FIG's course. Contact these board members and let them know where FIG should be going.

—John Hall, President, FIG

RETRY, EXIT, and Word-Level Factoring

Richard Astle
La Jolla, California

With the standardization process completed, and mainstream acceptance the goal, now is not, perhaps, the time to propose lexical extension, or a different style, even idiom, of programming. RETRY, the word I propose here, is not, however, a whim, a mere "here's something cute you can do, aren't I clever," like implementing a C-like *switch* in Forth. I don't mean that SWITCH: doesn't have its uses, only that since I found RETRY a decade ago, I have learned (tenderly at first, but then more and more passionately) to love it, and to rely on it to the extent that I use it far more often than its well-known brethren, REPEAT, UNTIL, and AGAIN.

RETRY (perhaps not its best name, but easy to type) is the opposite of EXIT. Where EXIT unconditionally terminates a colon definition regardless of nested IFs, WHILEs, and UNTILs—leaping, so-to-speak, to the semicolon and beyond—RETRY returns unconditionally to the colon, to start the word all over again. Like EXIT, RETRY can't be used within a DO ... LOOP, but that's another story.

EXIT

EXIT, taken by itself, is useful as a way to get out when the unexpected occurs, as in deeply nested error handling. It can also be used to implement a crude but effective *case statement*, e.g.:

```
: TEST&ACT ( n --- )
  DUP n1 = IF DROP do-something1 EXIT THEN
  DUP n2 = IF DROP do-something2 EXIT THEN
  DUP n3 = IF DROP do-something3 EXIT THEN
  DUP n4 = IF DROP do-something4 EXIT THEN
  DUP n5 = IF DROP do-something5 EXIT THEN
  do-default-something
;
```

It is not insignificant that this structure not only looks better (perhaps a matter of opinion) but also saves space, at least in Forth implementations I've inspected. Of the following,

```
: DOG test-something?
    IF do-something EXIT
    THEN do-something-else ;

: DOG test-something?
    IF do-something
    ELSE do-something-else THEN ;
```

the first is two bytes shorter when ELSE compiles four bytes (a jump and an address) against EXIT's two. Of course, the shorter case is also uglier, unless the test distinguishes not merely between two things but two kinds of things, as between normal and error processing:

```
: DOG error?
    IF .error-message EXIT
    THEN do-something ;
```

or

```
: DOG error?
    -IF do-something EXIT
    THEN .error-message ;
```

(-IF is equivalent to 0= IF. I also have a -WHILE and a -UNTIL, but for me they are less useful.)

RETRY

RETRY, taken by itself, is a way to start over when execution goes awry, as when a user inputs an incorrect value, or an operating system or hardware device returns a message meaning "not ready." You can test for any number of errors and retry from each one, rather than piling up flags and ORing them together for a WHILE or an UNTIL. (There have been suggestions to allow multiple WHILEs between BEGIN and REPEAT, but those are typically complications, while RETRY and EXIT simplify.)

```
: DOG
    test1 IF RETRY THEN
    test2 IF RETRY THEN
    do-something ;
```

instead of

```
: DOG
    BEGIN
        test1 test2 OR 0=
    UNTIL
    do-something ;
```

Admittedly, this is not a particularly compelling example, but the RETRY version is arguably clearer in concept. It also avoids the problem that, in the second case, test2 is executed whatever the result of test1, which might waste time or cause other problems. To avoid that situation, DOG could be recoded:

```
: DOG
    BEGIN
        test1 DUP 0=
        IF DROP test2 THEN
        0=
    UNTIL
    do-something ;
```

or perhaps

```
: DOG
    BEGIN
        test1 DUP
        -IF DROP test2 THEN
    -UNTIL
    do-something ;
```

but whatever is gained by this is not readability.

Taken together, RETRY and EXIT can untangle even more complicated control situations. Figure One shows a contrived, but far from extreme, example of RETRY and EXIT working together:

Word-Level Factoring

What RETRY and EXIT, taken together, make possible is flexible control structures with the word as the unit. This is, of course, a virtue of necessity: if RETRY (or 0= IF RETRY THEN) is like a REPEAT or UNTIL to an implicit BEGIN, that BEGIN can only be at the beginning of the word, with nothing between it and the word's name; similarly, the implicit REPEAT that might be seen as EXIT's target can have nothing between it and the semicolon. But this necessity promotes a way of factoring that is not at all unpleasant. The word becomes, even more than usually, a little machine, a building block, a subassembly, a white box. In a (typical) Forth word that begins with some action (setting up parameters for a loop, for example), then executes a REPEAT or UNTIL loop, and finally interprets and/or cleans up the results of the loop, the loop can be factored as a separate word. The enclosing word then executes straight through, and the factored word contains and controls the complexity, with opportunities for structural simplification by RETRY and EXIT. Nested structures can iteratively be factored away. In the extreme, one can do away with BEGIN, WHILE, REPEAT, UNTIL, and, perhaps, even ELSE entirely, factor all control structures (except for LOOPs) as separate colon definitions, and have that thing we all secretly desire, an odd-looking Forth.

Implementation

RETRY can be implemented through manipulation of the Forth instruction pointer (like REPEAT, UNTIL, and AGAIN) or the return stack (like RECURSE). The latter was the first method I tried, and the following implementation still exists (occasionally under the name >RETRY) in some derivatives of Guy Kelly's Forth-83:

```
: (RETRY)
    R> PERFORM ;

: RETRY
    COMPILE (RETRY) [COMPILE] RECURSE
; IMMEDIATE
```

(PERFORM is equivalent to @ EXECUTE.) This implementation has the virtue of conceptual difficulty, as does anything that involves twisting the return stack, and made me feel like a wizard when I found it. Basically, this implementation of RETRY, in code like the following,

```
: DOG ... RETRY ... ;
```

lays down the CFAs of (RETRY) and DOG. When the Forth interpreter goes off to execute the compiled (RETRY), it first places the address of the following cell (the one containing the recursive call to DOG) on the return stack, which R> PERFORM then executes, without any return stack buildup. Of course, this depends on the fact that the Forth this implementation works in is a single-segment Forth (or at least that data and lists are in the same segment), and that the instruction pointer is post-incremented; but a similar technique should be possible in other Forth implementation models.

Much more straightforward is the following, which should work in many 83-standard implementations, albeit using some not quite standard words.

```
: RETRY
    LATEST NAME> >BODY
    COMPILE BRANCH <RESOLVE
; IMMEDIATE
```

Figure One. Untangling complex control situations.

```
: DOG
    do-something       test1?    IF .error EXIT THEN
    do-something-else  test2?    IF RETRY       THEN
    do-some-other-thing ;
```

replacing

```
: DOG
  BEGIN
    do-something  test1?
    IF .error FALSE TRUE
    ELSE do-something-else test2? 0= ?DUP
    THEN
  UNTIL
  IF do-some-other-thing THEN ;
```

The phrase `COMPILE BRANCH <RESOLVE` is the code for `AGAIN` in the absence of "compiler security." (`AGAIN` is, of course, not quite 83-standard but often present, and it returns in ANS Forth.) In a Forth with compiler security, something like the following might work:

```
: RETRY
    LATEST NAME> >BODY
    1 [COMPILE] AGAIN
; IMMEDIATE
```

where 1 is the value placed on the stack at compile time by `BEGIN`.

In LMI's WinForth, the following definitions work:

```
: RETRY
    ?COMP LAST @ N>BODY
      1 [COMPILE] AGAIN
; IMMEDIATE

: RETRY
    ?COMP LAST @ N>BODY
      COMPILE branch    HERE - ,
; IMMEDIATE
```

All of these latter implementations are based on the same basic idea: to lay down the code for an unconditional backwards jump (an `AGAIN` or `FALSE UNTIL`) to an implicit `BEGIN` at the beginning of the colon definition (with, of course, the difference that you don't have to worry about nesting and balancing and all those things).

Richard Astle has been programming in Forth for about eight years, most of that time developing and maintaining a rather large database-management set of programs. In the process, he has re-written the underlying Forth system more than once for speed and capacity. He has a bachelors degree in mathematics 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).

Total control with *LMI FORTH*™

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

For Development:
Interactive Forth-83 Interpreter/Compilers for MS-DOS, 80386 32-bit protected mode, and Microsoft Windows™
- Editor and assembler included
- Uses standard operating system files
- 500 page manual written in plain English
- Support for graphics, floating point, native code generation

For Applications: Forth-83 Metacompiler
- Unique table-driven multi-pass Forth compiler
- Compiles compact ROMable or disk-based applications
- Excellent error handling
- Produces headerless code, compiles from intermediate states, and performs conditional compilation
- Cross-compiles to 8080, Z-80, 64180, 680X0 family, 80X86 family, 80X96/97 family, 8051/31 family, 6303, 6809, 68HC11
- No license fee or royalty for compiled applications

Laboratory Microsystems Incorporated
Post Office Box 10430, Marina Del Rey, CA 90295
Phone Credit Card Orders to: (310) 306-7412
Fax: (310) 301-0761

MAKE YOUR SMALL COMPUTER
THINK BIG
(We've been doing it since 1977 for IBM PC, XT, AT, PS2, and TRS-80 models 1, 3, 4 & 4P.)

FOR THE OFFICE — Simplify and speed your work with our outstanding word processing, database handlers, and general ledger software. They are easy to use, powerful, with executive-look print-outs, reasonable site license costs and comfortable, reliable support. Ralph K. Andrist, author/historian, says: "**FORTHWRITE** lets me concentrate on my manuscript, not the computer." Stewart Johnson, Boston Mailing Co., says: "We use **DATAHANDLER-PLUS** because it's the best we've seen."

MMSFORTH System Disk	from $179.95

Modular pricing — Integrate with System Disk only what you need:

FORTHWRITE - Wordprocessor	$99.95
DATAHANDLER - Database	$59.95
DATAHANDLER-PLUS - Database	$99.95
FORTHCOM - for Communications	$49.95
GENERAL LEDGER - Accounting System	$250.00

MMSFORTH

MILLER MICROCOMPUTER SERVICES
61 Lake Shore Road, Natick, MA 01760
(508/653-6136, 9 am - 9 pm)

FOR PROGRAMMERS — Build programs FASTER and SMALLER with our "Intelligent" MMSFORTH System and applications modules, plus the famous MMSFORTH continuing support. Most modules include source code. Ferren MacIntyre, oceanographer, says: "Forth is the language that microcomputers were invented to run."

SOFTWARE MANUFACTURERS — Efficient software tools save time and money. MMSFORTH's flexibility, compactness and speed have resulted in better products in less time for a wide range of software developers including Ashton-Tate, Excalibur Technologies, Lindbergh Systems, Lockheed Missile and Space Division, and NASA-Goddard.

MMSFORTH V2.4 System Disk from $179.95
Needs only 24K RAM compared to 100K for BASIC, C, Pascal and others. Convert your computer into a Forth virtual machine with sophisticated Forth editor and related tools. This can result in 4 to 10 times greater productivity.

Modular pricing — Integrate with System Disk only what you need:

EXPERT-2 - Expert System Development	$69.95
FORTHCOM - Flexible data transfer	$49.95
UTILITIES - Graphics, 8087 support and other facilities.	

and a little more!

THIRTY-DAY FREE OFFER — **Free MMSFORTH GAMES DISK** worth $39.95, with purchase of MMSFORTH System. CRYPTOQUOTE HELPER, OTHELLO, BREAKFORTH and others.

Call for free brochure, technical info or pricing details.

Making Forth Professional

Peter Knaggs
Paisley, Scotland

In the past, Forth has suffered (and continues to suffer) from the attitude that it is a "hackers" language. This goes back to the release of fig-Forth—software houses/managers took one look at this hodge-podge of ideas and badly documented pre-standard code before either becoming engrossed in the idea or dropping it like a hot brick. Now when we talk of Forth, this is what they remember. We are well aware that Forth has moved on greatly since those days, yet this is all they remember.

Software managers tend to see only the (mostly badly documented) code. They do not appreciate that Forth is not just another programming language, but rather a philosophy of programming. Many ideas currently in favour (such as structured programming, reusability, libraries, etc.) generally known as "software engineering" have been available and used in Forth for many years.

This must be changed, if Forth is to gain any standing as a language. Fortunately, this has indeed been happening, the ANS Forth being a very big step in this direction, closely followed by the IEEE Open Firmware standard. These two events are slowly bringing some respectability to Forth.

If Forth is to be taken seriously, it should have a professional body that controls the quality of Forth programmers.

A number of platform-independent development languages have been developed by different organisations. The Ten15 development language (TDL), architecture-neutral distribution format (ANDF), Open Firmware, and JAVA are all languages that use a Forth-like abstract machine to provide "portable programming" at various levels.

The ANSI standard has gone a long way to improving Forth's standing in many manager's eyes. The "portability" aspect of the standard is very important; the standard not only addresses this problem, but actively encourages the development of portable programs!

The professional acceptance of Forth is being held back by both the lack of good quality Forth programmers and the reluctance of companies to acknowledge their use of Forth.

The Open Firmware standard has created a need for professional-quality Forth programmers which we are simply not able to satisfy.

A growing number of companies have been using Forth but are reluctant to acknowledge it, normally for one of two reasons: they either consider it to be commercially sensitive or their software managers think they will be condemned for making what was, apparently, an intelligent production decision.

The academic interest in Forth, both as a language in its own right and as a development platform, is being hindered by the push for ever-more refereed papers. There is currently very little scope for "academic publication," because we only offer two outlets for this kind of work.

The euroFORTH conference has recently introduced a "refereed section" to its proceedings. None of the other Forth meetings support refereed publications. The only other outlet being the *Journal of Forth Application and Research,* which has only recently started production again after a gap of five years. The other Forth-related publications are not refereed.

This is a vast improvement over the situation in '93 when nothing was available. However, there is still a long way to go if Forth is to be accepted in academia.

It is my belief that fig-Forth, or more precisely, the non-updating of fig-Forth, has been the cause of much of the predicament we find ourselves in today, in that both the Forth Interest Group (FIG) and Forth are permanently associated with the "hackers" attitude.

During the 1991 Rochester Forth Conference, I was a party to a number of discussions with various people both at FIG (including the chairman and directors) and the Institute of Applied Forth Research (IAFR). I have reflected on these discussions and have come up with the following suggestion.

If we want Forth to be taken seriously, we should have a professional body/organisation that controls the quality of Forth programmers. This is a very important part of the thinking, as only a via a professional body can we overcome the hacker label. For the time being, I will call this new organisation the "Forth Programmers Guild."

I see the Guild as a membership-based organisation to monitor and promote good practise in and the further development of the Forth programming environment. The exact nature of the Guild is unknown; however, I see it having a number of roles in order to fulfill the above goal:

- The Guild would be subscription based, with different levels of membership based on the person's competence in professional software development with Forth. This would confer an official status on the member, such as "Student," "Associate," "Member," or "Fellow."

  There will be a clear set of criteria specifying the level of competence for the different levels of membership. This would give members a clear indication of which areas they should improve in order to progress up the membership scale.

  It would also give employers a clear understanding of the person they are employing and their abilities.

- There are a number of companies currently offering courses in Forth programming. It is envisaged that, in time, the Guild would validate these courses. This would allow people taking any such course to become a student member of the Guild and, on completion of the course (or courses), sufficient experience to become a member.

  This would allow the companies offering such courses to advertise it as being validated by the Guild, thus raising the value of the course and the profile of the Guild. It would also encourage membership in the Guild.

- To encourage further development of the Forth programming environment, the Guild would organise a number of international, refereed conferences. It could also arrange so-called "picnics" at a national or regional level for social contacts.

  It should also provide a medium for more-considered works, thus it should publish a refereed journal on Forth Development, Practise, and Experience. The frequency of this is unknown, possibly as low as one issue per year. On the other hand, as with the picnic idea, it should also produce a more lightweight newsletter on a more regular basis (say monthly) with news, articles, jobs, etc. covering the more social aspect of things.

As we can see, the proposed Guild is going to be rather large and encompass the current operation of both FIG and IAFR. Precisely how the Guild is going to be set up is another question.

One possibility would be to start a entirely new organisation. However, this would be in direct competition with both FIG and IAFR. Whilst this would be the correct "free market" approach, it will simply fragment the community further. It would lead to a great deal of duplicated effort for a while, until one or more of the three players retired. Not a happy picture.

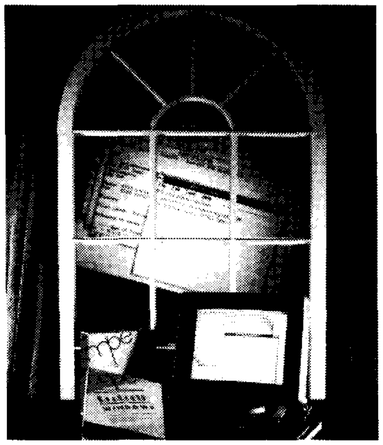
Another possibility would be to set up a special interest group under the control of one of the "professional" organisations, such as the BCS, ACM, IEE, or even the IEEE. This reminds me very much of the now-defunct ACM SIG-Forth, except they were not as active. There is a possibility of some funding from the Open Firmware camp for such a move.

Alternatively, either FIG or IAFR may expand their operations to become the new "Guild."

A final possibility would be for FIG and IAFR to merge into the new Guild. This still has a drawback: the "hackers" attitude attached to FIG may move over to the new organisation, thus it would make more sense for the IAFR to take over FIG.

The ideas I have presented here are open to discussion. Some of the ideas are covered in more detail in my euroFORTH '93 paper, "A Look at Forth's Academic Standing" also available for ftp at <URL:ftp://ftp.paisley.ac.uk/pub/cis/forth/ef93/academic.ps> or on the World Wide Web at <URL: http://www.paisley.ac.uk/~cis/forth/ef93/academic.html>.

Peter Knaggs is a lecturer in Software Engineering at the University of Paisley, Scotland. He obtained his Ph.D. with a thesis entitled "Practical and Theoretical Aspects of Forth Software Development" from the University of Teesside in 1994. He is actively involved in Forth-related research and is the program chair for the European Forth conference (euroFORTH), and administers the euroFORTH WWW site <URL: http://www.paisley.ac.uk/~cis/forth>.



At last...

ProForth for Windows

...brings the full power of Forth to Windows!

- Powerful 32-bit Forth for Windows and NT.
- Includes ProForth GUIDE™ "visual"-type automated toolkit for Windows user interfaces.
- Graphics library, floating point, much more.
- Full support for DDE, external DLLs.
- Integrated debugging aids for reliable programs.

Go with the systems the pros use... Call us today!

FORTH, Inc.
111 N. Sepulveda Blvd, #300
Manhattan Beach, CA 90266
800-55-FORTH 310-372-8493
FAX 310-318-7130 forthsales@forth.com

ProForth for Windows is a product of Microprocessor Engineering Ltd. (MPE), Southampton, England. ProForth for Windows is sold and supported in the US and Canada by FORTH, Inc.

1995 Rochester Forth Conference

Emerging Technology

Nicholas Solntseff
Hamilton, Ontario, Canada

Driving down to Rochester for three hours on a hot summer afternoon gave me the opportunity to reflect on the future of the Forth Interest Group (FIG), my own involvement in Forth, and where personal computing is heading. The fastest growing aspect of the latter in the Hamilton-Wentworth region, as in most Canadian communities, is the rapid growth of community networks, or *freenets* as they are commonly called. The last meeting of the Hamilton Internet Users Group collected some 40 keen persons eager to find out what they could about the mysteries of the Internet. By contrast, the June meeting of the South Ontario Chapter of FIG attracted only four in addition to the speaker. The highest attendance ever was 30, ten years ago. Perhaps something should be done about combining Forth and the Internet!

This year, the attendance at the fifteenth Rochester Forth Conference was 46, which showed, perhaps, that Forth in an industrial setting is withstanding the onslaught of C and C++. The vendors (Forth Inc., New Micros, and MPE) were in an upbeat mood and talked of profits at long last, the registrants expressed hope that Open Boot Forth would lead to more neophytes at future meetings, while the half-dozen or so attendees for whom this was their first contact with the Forth culture eagerly described their successes in taming Forth.

...we need a "killer app" to demonstrate that Forth is the best solution to a pressing problem...

Registration and tutorials took place Wednesday afternoon; papers were presented Thursday and Friday, with working groups taking place Friday afternoon. A Forth Open Day with more tutorials and demonstrations by vendors was held on Saturday. As usual, food was good, free beer plentiful, and evening discussions wide ranging and stimulating. An interesting change from the 12 other times I attended the conference was the large numbers of conferees from two other meetings being held at the same time—the North-East Synod of the Presbyterian Church and the American Electrostatic Society. The dining room was very crowded at breakfast time, and I could imagine what it would be like for Forth to be part of mainstream computing—conferences attended by several thousand people!

The proceedings were opened Wednesday morning by Larry Forsley, who pointed out that programming skills among computer users were diminishing while more and more capability was being put into silicon and proprietary packages that nobody could talk about. What was needed was to reduce the average age of attendees at future Rochester Conferences by attracting young, new blood. Forth programming instruction geared to those encountering Forth for the first time in Open Boot courses was a possible way of achieving this.

Chuck Moore gave the first invited talk, and discussed his involvement with chip making. There are many opportunities for innovative engineering apart from making bigger and bigger chips. Chuck's latest designs have an area around one-tenth that of the giant Pentium chips and, in fact, involve a volume occupied by some 3000 x 3000 x 60 atoms, which is getting pretty close to the nanotechnology scale. Designing chips is fun, but they have to be manufactured, which puts immense pressure on the designer to meet foundry schedules; and tested, which brings about a very long, drawn out, edit-compile-test cycle with a period of one month! How to improve this is the challenge for the next century.

Thirty-three general papers were presented on the usual wide variety of topics, although papers on astronomical applications were notable absent. The largest object discussed at this year's conference was the size of a football field—a lime kiln controlled by Alan Anway's Forth system. Four papers from the Electrical Engineering Department at the University of Missouri-Rolla by Darrow F. Dawson et al. dealt with the use of Forth in a classroom environment and described the hardware (a single-board computer based on the TMS320C25 DSP), the software (a Forth and a graphical user interface for Unix), as well as a set of student exercises. Richard E. Haskell (CSE Department, Oakland University) detailed a C++-based version of Forth for the Motorola 68HC11 designed to reside in embedded systems.

Commercial applications described at the conference highlighted interoperability requirements, i.e., the need for Forth systems to work in a wide variety of environments, with multi-vendor hardware and software packages, as well as in a multi-processing regime. This means that more and more effort has to be devoted to the interaction between components rather than the individual components themselves. The use of standard modules such as those for X-Windows, TCP/IP, and operating systems is becoming increasingly necessary. An important aspect of marketing Forth is to concentrate on selling solutions, not tools. Forth allows the vendor to demonstrate a prototype in very short time by comparison with other approaches, and this can lead to user satisfaction and trust in the developer without the need to mention Forth at all!

The program was rounded out by several papers exploring new directions for Forth internals and applications—these included a Forth-based graphical process-chart language (Matthew Mercaldo), a new way of looking at stack operations (Rob Chapman), linked-list techniques (Lloyd Prentice), the use of Field Programmable Gate Arrays (David Rusnell), a Forth-based approach to processing SGML and HTML documents (Norman Smith), to name just a few. The Forth Scientific Library managed by Skip Carter has been the success of the year! It has the potential to provide users outside the present Forth community with an alternative to FORTRAN for their computing needs. The library is steadily expanding, but more help is needed to review contributed software. It can be found at <URL:http://taygeta.oc.nps.navy.mil/fsl/sciforth.html>.

Whether freeware and cheapware Forths hurt Forth vendors or eventually provide them with new customers was debated, as expected, during social periods outside the lecture room and in dormitory corridors. Proponents and opponents of an under-$40 Forth went home without changing their minds, but I have decided that what we need is a "killer app" that will demonstrate unequivocally that Forth is the best solution to a pressing problem. How about an interactive scripting package for the World Wide Web?

Nicholas Solntseff, a professor in the Department of Computer Science and Systems at McMaster University (Hamilton, Ontario, Canada) is a board member of the Forth Interest Group. He can be reached via e-mail at ns@maccs.dcss.mcmaster.ca.

Statement of Ownership, Management, and Circulation

1. Publication Title: *Forth Dimensions*
2. Publication No. 0002-191
3. Filing Date: Sept. 27, 1995
4. Issue Frequency: bi-monthly
5. No. of Issues Published Annually: 6
6. Annual Subscription Price: $40
7. Complete mailing address of known office of publication: P.O. Box 2154, Oakland, California 94621-0054
8. Complete mailing address of headquarters or general business office of Publisher: same as above
9. Publisher: Forth Interest Group, P.O. Box 2154, Oakland, California 94621-0054. Editor: P.O. Box 2154, Oakland, California 94621-0054.
10. Owner: Forth Interest Group (non-profit), P.O. Box 2154, Oakland, California 94621-0054.
11. Known bondholders, mortgagees, and other security holders owning or holding 1 percent or more of total amount of bonds, mortgages, or other securities: none.
12. The purpose, function, and nonprofit status of this organization and the exempt status for federal income tax purposes: has not changed during preceding 12 months.
13. Publication Name: *Forth Dimensions*
14. Issue Date for Circulation Data Below: Nov.-Dec. 1995

15. Extent and Nature of Circulation:	*Average No. Copies Each Issue During Preceding 12 Months*	*Actual No. Copies of Single Issue Published Nearest to Filing Date*
a. Total No. Copies (*net press run*)	1300	1300
b. Paid and/or Requested Circulation		
(1) Sales through Dealers and Carriers, Street Vendors, and Counter Sales (*not mailed*)	0	0
(2) Paid or Requested Mail Subscriptions (*Include Advertisers' Proof Copies/Exchange Copies*)	1082	1061
c. Total Paid and/or Requested Circulation	1082	1061
d. Free Distribution by Mail (*Samples, Complimentary, and Other Free*)	6	6
e. Free Distribution Outside the Mail (*Carriers or Other Means*)	0	0
f. Total Free Distribution	6	6
g. Total Distribution	1088	1067
h. Copies Not Distributed		
(1) Office Use, Leftovers, Spoiled	212	233
(2) Return from News Agents	0	0
i. Total	1300	1300
Percent Paid and/or Requested Circulation	99.4%	99.4%

Signature and Title of Editor, Publisher, Business Manager, or Owner: John D. Hall, President, September 27, 1995.

F-PC

Nanocomputer Optimizing Target Compiler The PIC16C71 and PIC16C84 Library

Tim Hendtlass
Hawthorn, Victoria, Australia

[Part one of this article, pertaining to the processor-independent core, appeared in our preceding issue.—Ed.]

This example of a processor-specific library is for a particular pair of nanocomputers which share substantially the same instruction set, but have different hardware resources. It is presented as a model for readers who wish to develop versions of NOTC for other processors.[1] First the hardware must be studied, and the way to implement the stacks decided. It is often advantageous to keep the top of the data stack in a register. Depending on the architecture, the one stack may handle both return address and control information (as in traditional Forth), or separate return and control stacks may be needed. The boot-code and the init-code will need to be designed.

The PIC16C71 and PIC16C84 chips are similar; the C71 has a four-channel analogue-to-digital converter, and the C84 has EEPROM rather than EPROM. Both employ Harvard architecture, in which the program memory and the data memory are quite separate and there is no way to transfer information between the two. The program memory is 14 bits wide, the data memory is eight bits wide. Each chip has a processor that handles 14-bit instructions, 1024 14-bit words in which to store the program, twelve input/output lines, and an independent timer. There is a small amount of eight-bit-wide RAM, but only locations 12 to 47 are available to us—the rest are used to control the other hardware on the chip or for processor registers. A separate 14-bit wide, 8-deep hardware stack is used for holding return addresses.

Since the return stack is unavailable for holding control information, and it is really not practical to hold that information on the data stack, a third stack is needed if control structures are to be used. The normal processor return stack is used for return information and the other two stacks, data and control, have to share the limited, byte-wide RAM along with any variables we may construct. Basic variables, like all math operations, are limited to eight bits.

The organization of the memory is shown in Table One.

There is one special register associated with the processor, the W register, that always supplies one parameter to the arithmetic logic unit. The result of an operation can either be returned to the W register or to the other register involved in the operation. The W register does not have an address in normal RAM, and is a very busy register. Also, location four is special: it is used as a pointer into the rest of RAM. Any read or write to address zero will actually occur to the address which is held in location four. This makes location four ideal for maintaining a stack in RAM; unfortunately, it has to be shared between the data and control stacks. We leave it normally containing the data pointer, switching it with the control pointer as required.

Initially, the processor will have to be set so that the control-stack pointer contains 0Fhex (remember, it will be incremented before anything is stored on it). The data stack pointer will point to the lowest data memory address occupied by any variables, and it will be decremented before any item is stored onto the data stack.

The number of words that have to be defined in machine code is small and is taken from the eForth primitive set. As this is a true compile environment, words that are only used in the interpret mode are omitted. However, a few extra words are defined in the interests of speed. The standard Forth words defined are:

Data-stack-manipulation operations:
`DUP, NIP, DROP, SWAP, OVER, ROT`

Arithmetic and logical words:
`+, -, UM+, AND, OR, XOR, NOT`

Comparison words:
`=, 0=, 0>`

Control-stack operations:
`R>, >R`

As noted above, some of these words are additional to the minimum eForth set—for example, + and -, which can be derived from UM+. However, they are written in machine code for speed.

Further speed improvements could be made by writing other words in machine code (for example, *) and removing the corresponding high-level definitions in the processor-independent core. Some common routines (not standard Forth words) used in the above definitions have

1. The code produced by the compiler using this library has only been tested on a simulator. Users do so at their own risk!

Table One. Memory organization of the PIC chips.

Address	Symbolic name	Use	
W register	D1	Top of the data stack	
0	Stack	Access location pointed to by DPTR (or CPTR if pointers swapped)	
1–3		Other processor registers	
4	Stack pointer (SP)	Pointer to second item on data stack (D2)	
5–12		Other processor registers	
13	Control pointer (CP)	Alternate stack pointer, normally pointer to top item on control stack	
14	Temp1	Working space	
15	Temp2	Working space	
16	Cn	Bottom item on control stack of x items	Control stack ↓
17	Cn-1	next to bottom item on control stack	
		....	
16+x		Top item on control stack	
		Unused space	
		Top item on data stack	Data Stack ↑
		....	
		Bottom item on data stack	
47–n		Last (nth) declared variable	Variables ↕
		....	
47		First declared variable of n	
128–139		processor registers	

been identified and coded as subroutines. These are:

`POP`	Remove the top item from the data stack putting it in temporary register Temp1.
`PUSHT1`	Move item from Temp1 to top-of-stack.
`PUSHT2`	Move item from Temp2 to top-of-stack.
`SHUFFLE`	Top-of-stack to Temp2, Temp1 to top-of-stack.
`CSPtoSP`	Save normal stack pointer and move control-stack pointer into normal stack-pointer place.
`SPtoCSP`	Return item in normal stack pointer to control-stack pointer, replace saved stack pointer.

Since the nanocomputers have two I/O ports, simple specific words are provided to read and write to these ports (`C5@`, `C5!`, `C6@`, `C6!`, `C85@`, `C85!`, `C86@`, `C86!`). It did not seem worthwhile to implement the conventional `C!` and `C@` words. (Location 85 is the data direction register for real port 5; location 86 is the data direction register for real port 6.)

The definitions in the library are stored as in-line routines; if necessary, they will be converted to a subroutine either by adding a return instruction (0008hex) or by changing the final call to a goto. All library entries need to have their breakeven number specified; if they are used more than this number of times, it will be more efficient to load them as a subroutine and call them as needed. If the breakeven count is set to zero, they will always be loaded as a subroutine; if set to a very high number, such as 7FFFhex, they will always be used in in-line form.

Table Two shows the breakeven number as a function of the length of the in-line form of the routine and whether the in-line form ends in a call.

The source code published with this section of the article is organised in parts. First, a number of definitions are provided that were left as deferred definitions in the core. Then words are defined which will, when run, lay down specific processor instructions in the image. The names given reflect the use being made of various registers on the processor which, hopefully, makes it all more readable than using normal processor assembly mnemonics. The processor-dependent primary words are then defined, followed by three flow-control primitive words from which all the normal control structures are built. Next come words to lay down in-line numbers and to remove numbers laid down that further consideration has shown are not needed here (see how

Table Two. Calculating in-line vs. subroutine efficiency.

Length (as in-line code)	1	2 no end call	2 ends in call	3 no end call	3 ends in call	>3
Load as subroutine if used more than...	7FFFhex (never)	3	2	2	1	1

Sample output.

The particular test file was:

```
\ file to test compiler operation- actual program makes no sense!
    variable fred
    variable paul
  5 constant joe
: Word1 if dup 5 else dup paul nip then rot 20 rot ;  \ a comment
: Word2 begin swap swap while swap swap word1 repeat fred word1 ;
: Word3 begin joe over >r ( a comment ) swap word2 r> until ;
```

Sample output during the compile session:

```
compile test of 311 bytes.
pass 1 ******* Image length now 2
pass 2 ************************************ Image length now 44
pass 3 ******* Image length now 100
Final image size = 100 words ok
```

Output obtained by using *print-out*:

```
Library usage
DUP                 used 6 times loaded as subroutine at 2
DROP                used 3 times loaded inline
POP                 used 2 times loaded inline
PUSHT1              used 1 times loaded inline
PUSHT2              used 2 times loaded inline
SHUFFLE             used 1 times loaded inline
SWAP                used 7 times loaded as subroutine at 5
OVER                used 1 times loaded inline
NIP                 used 1 times loaded inline
CSPTOSP             used 2 times loaded as subroutine at 13
SPTOCSP             used 2 times loaded as subroutine at 18
>R                  used 2 times loaded as subroutine at 23
R>                  used 2 times loaded as subroutine at 32
ROT                 used 2 times loaded as subroutine at 40
FRED                used 2 times loaded inline
PAUL                used 2 times loaded inline
5                   used 2 times loaded inline
20                  used 1 times loaded inline

Symbol table
WORD3               80      [  50]
WORD2               61      [  3D]
WORD1               43      [  2B]
ROT                 40      [  28]
R>                  32      [  20]
>R                  23      [  17]
SPTOCSP             18      [  12]
CSPTOSP             13      [   D]
SWAP         5      [   5]
DUP                 2       [   2]
ISPACE              10316   [284C]

Memory Map
Address             Contents
   0.[   0]....[284C][   8][ 384][  80][   8][  8E][ 800][ A84]
   8.[   8]....[  8F][ 80E][2002][ 80F][   8]{ 804][  8E][ 80D]
  16.[  10]....[  84][   8][ 804][  8D][ 80E][  84][   8][  8F]
  24.[  18]....[200D][ A84][ 80F][  80][2012][ 800][ A84][   8]
  32.[  20]....[2002][200D][ 800][ 384][  8F][2012][ 80F][   8]
  40.[  28]....[2017][2005][2020][  8E][ 800][ A84][ 88E][1D03]
  48.[  30]....[2835][2002][2002][3005][2838][2002][2002][302E]
  56.[  38]....[ A84][2028][2002][3014][2828][2005][2005][  8E]
  64.[  40]....[ 800][ A84][ 88E][1903][2849][2005][2005][202B]
  72.[  48]....[283D][2002][302F][282B][300F][ 10C][302D][ 104]
  80.[  50]....[2002][3005][  8E][ 800][ A84][  8F][2002][ 80E]
  88.[  58]....[2002][ 80F][2017][2005][203D][2020][  8E][ 800]
  96.[  60]....[ A84][ 88E][1D03][2850]
```

constants are handled). A word to allocate space for a variable is followed by one to extract the image in a form suitable for transferring it to the actual target processor (here left as a dummy definition).

The hardest part of writing the library is ensuring that you use the correct version of words. Most of the library is written using the special definitions in the library itself, but from time to time the normal control structure words (IF, etc.) and compiling words like : from the FORTH vocabulary are needed. Accidentally running the library versions that lay down code in the target when you meant to use the "real" definitions is guaranteed to lock up your computer! The vocabulary search order is set up to suit the majority of words in the definition being compiled, but the order will have to be temporarily changed for the minority words. For example, the search order might be ROOT FORTH NOTC LIBRARY. Words will be searched for in the LIBRARY first; only if they are not found there will NOTC and then FORTH be searched. If we need the FORTH version of >R (for example), rather than the version in the library that would lay down code in the target image, we need to add FORTH to the top of the search list, find the >R, and then remove FORTH from the top of the list again. The sequence to do this is [ALSO FORTH] >R [PREVIOUS].

Tim is a long-time Forth devotee. A professor in the School of Biophysical Sciences and Electrical Engineering at Swinburne University, he spends much of his time working with artificial neural networks and evolutionary algorithms in his capacity as Director of the Centre for Intelligent Systems. He escapes to Forth whenever he gets the chance, and has a dream of building a giant evolutionary algorithm computation array using Forth chips. He can be contacted at Swinburne (P.O. Box 218, Hawthorn 3122 Australia) or by e-mail (tim@bsee.swin.edu.au). The source code for this article can be obtained by anonymous ftp from brain.physics.swin.oz.au in pub/forth, as can electronic copies of his book on F-PC, *Real Time Forth*.

Source code — PIC library.

```
\ Nanocomputer Optimising Target Compiler. (NOTC) PIC16C71 and C84 library. ver 1.0
\ first few definitions go to NOTC
  only forth also notc          \ search order notc>forth>root
  also definitions              \ definitions to NOTC

\ ***************************************************VARIABLES AND CONSTANTS
  VARIABLE TDS 47 TDS !         \ points to absolute top-of-data-stack area
  1024 CONSTANT max_program     \ allow target to have up to 1024 program steps
  2 CONSTANT cell_size          \ 2 bytes needed for each 14-bit instruction
\ ***********************************************************************

\ make space for the image in the image vocabulary
  only forth also notc also target \ search order target>notc>forth>root
  also definitions                 \ definitions to target
  CREATE ISPACE max_program cell_size * allot   \ build the array

\ now start adding definitions to the library
  only forth also notc also target also library
  also definitions              \ definitions to library

\ WORDS TO MANIPULATE THE IMAGE

\ ****************************** THE DEFERRED WORDS FROM CORE ARE DEFINED HERE

\ the primitive words to read and write to image space
: I! ( n -- )                   \ write one 16-bit cell
  Iptr @ cell_size * Ispace + !
;
: I@  ( -- n )                  \ read one 16-bit cell
  Iptr @ cell_size * Ispace + @
;

: <IADD> ( n -- )               \ add an item into image space
  Iptr @ max_program =          \ hit maximum size ?
  if 3 error                    \ yes, abort
  else 1 Iptr +! I!             \ no, lay down next 16-bit cell
  then
;
: IADD ( item -- )              \ add item into image unless in pass 1
  p1?                           \ if pass 1 ..
  if drop                       \ lose item
  else <iadd>                   \ otherwise add it
  then
;

\ **************************************************HANDLING CALLS AND RETURNS

: <ICALL> ( n -- )
   2000H or Iadd                \ lay down call n
;
: RETURN   0008H Iadd ;         \ lay down machine code return
\ If last cell is a call change it to a goto (jump), else add return (0008H)
: <IRETURN> ( -- )
  I@ 3800H and 2000H =          \ result will be true if last cell is a call
  if I@ 0800H or I!             \ if call make it goto
  else return                   \ otherwise add return
  then
;
\ Complete the deferred definitions from the core
' <Ireturn> is Ireturn
' <Icall> is Icall

\ *******************************************************PROCESSOR PRIMITIVES
\ These routines cause single machine instructions to be laid down during pass 2.
\ They have been given names that reflect their particular use here.
: D1>t1   008EH Iadd ;          \ MOVWF t1 - copy top-of-data-stack to temp1
: t1>D1   080EH Iadd ;          \ MOVF t1,0 - copy temp1 to top-of-data-stack
: D1>t2   008FH Iadd ;          \ MOVWF t2 - copy top-of-data-stack to temp2
: t2>D1   080FH Iadd ;          \ MOVF t2,0 - copy temp2 to top-of-data-stack
```

(Continues.)

```
: D2>t2    008FH Iadd ;           \ MOVWF t2 - copy D2 directly to temp2
: PUSHD1   0080H Iadd ;           \ MOVWF SP - push top-of-data-stack
: POPD2    0800H Iadd ;           \ MOVF 0,0 - pop second on data stack to top
: DPT-     0384H Iadd ;           \ DECF DPTR - decrement the data-stack pointer
: DPT+     0A84H Iadd ;           \ INCF DPTR - increment the data-stack pointer
: t1+D1    070EH Iadd ;           \ ADDWF t1,0 - add D1 and t1
: t1-D1    020EH Iadd ;           \ SUBWF t1,0 - subtract D1 from t1
: t1andD1  050EH Iadd ;           \ ANDWF t1,0 - and D1 and t1
: t1orD1   040EH Iadd ;           \ IORWF t1,0 - or D1 and t1
: t1xorD1  060EH Iadd ;           \ XORWF t1,0 - xor D1 and t1
: 0>D1     3000H Iadd ;           \ MOVLW 0 - set D1 to zero
: 1>D1     3001H Iadd ;           \ MOVLW 1 - set D1 to one
: -1>D1    30FFH Iadd ;           \ MOVLW FF - set D1 to -1

\ ***********************************TARGET-ARCHITECTURE-DEPENDENT ROUTINES
: BOOT-CODE                       \ the low memory initial code, always loaded
  -1 iptr ! 0 Iadd                \ write a nop at start of image, to be patched later
  return                          \ and a simple return for the interrupt vector
;
: INIT-CODE                       \ the high memory initial code
  Iptr @ dup 1+ 2800H or          \ assemble jump to next instruction
  0 Iptr ! I! Iptr !              \ write jump at zero and restore Iptr
  300FH Iadd                      \ MOVLW 15
  010CH Iadd                      \ MOVWF 13 initialise control pointer
  TDS @ 3000H or Iadd             \ MOVLW [TDS] adr of last byte used by variables
  0104H Iadd                      \ MOVWF 4 initialise data-stack pointer
;

\ *****************************************PROCESSOR-DEPENDENT-LIBRARY WORDS
\ (no early word may call a later one)
 3 LIB: DUP dpt- pushd1 ;         \ increment stack pointer, duplicate top item.
 3 LIB: DROP popd2 dpt+ ;         \ pop old d2 to top-of-stack, adjust pointer
 2 LIB: POP d1>t1 drop ;          \ pop d1 off into templ
 2 LIB: SDROP d1>t1 drop ;        \ pop old d1 into templ
 3 LIB: PUSHT1 dup t1>d1 ;        \ push from templ to top-of-data-stack
 3 LIB: PUSHT2 dup t2>d1 ;        \ push from temp2 to top-of-data-stack
 3 LIB: SHUFFLE d1>t2 t1>d1 ;
 1 LIB: SWAP pop shuffle pusht2 ; \ interchange d1 and d2 (alters templ and temp2)
 1 LIB: OVER pop d2>t2 pusht1 pusht2 ; \ copy old D2 on top of old D1
 7FFFH LIB: NIP dpt+ ;            \ just increment data pointer so old D2 lost
 3 LIB: + pop t1+d1  ;            \ save d1, old d2 to d1, add saved d1 to new d1
 3 LIB: - pop t1-d1  ;            \ save d1, old d2 to d1, subtract saved d1 from new d1
 3 LIB: AND pop t1andd1 ;         \ save d1, old d2 to d1, and saved d1 to new d1
 3 LIB: OR  pop t1ord1  ;         \ save d1, old d2 to d1, or saved d1 to new d1
 3 LIB: XOR pop t1xord1 ;         \ save d1, old d2 to d1, xor saved d1 to new d1
 1 LIB: NOT d1>t1 090EH Iadd ;    \ d1 to templ, then comp of templ back to d1
 1 LIB: CSPtoSP                   \ save stack pointer in templ, put ctrl-stack pointer
                                  \   where stack pointer usually is. W is destroyed.
        0804H Iadd                \ movf sp,0 - data-stack pointer to D1
        d1>t1                     \ and into templ
        080DH Iadd                \ movf csp,0 - control-stack pointer to D1
        0084H Iadd                \ movwf sp - and into normal data-stack pointer place
    ;
 1 LIB: SPtoCSP                   \ reverse operation of CSPtoSP
        0804H Iadd                \ CSP from normal DP place to D1
        008DH Iadd                \ movwf asp - CSP from D1 back to its usual place
        t1>d1                     \ SP back to D1
        0084H Iadd                \ and back to its usual place
    ;
 1 LIB: >R                        \ top item from data stack to control stack
        D1>t2                     \ top item to temp2. D1 now free.
        CSPtoSP                   \ swap to control stack
        DPT+                      \ adjust stack pointer to point to next space in
                                  \   control area
        t2>D1                     \ return top item to D1
        PUSHD1                    \ move into place
        SPtoCSP                   \ back to data stack
        DROP                      \ adjust data stack so top item is in D1
    ;
 1 LIB: R>                       \ top item from control stack to data stack
        DUP                      \ make room in D1
        CSPtoSP                  \ swap to control stack
        POPD2                    \ old top-of-control-stack to D1
        DPT-                     \ adjust control-stack pointer value
        D1>t2                    \ item to temp2
        SPtoCSP                  \ back to data stack
        t2>D1                    \ and put item in place on top-of-stack
      ;
 1 LIB: ROT
        >r swap r> swap          \ a b c - c a b
      ;
 1 LIB: UM+                      \ add D1 and D2, answer in D2, carry in D1
        +                        \ do the addition
        dup                      \ push answer, carry preserved
        0>D1                     \ clear D1, carry preserved
        1803H Iadd               \ btfsc 3,0, skip next instruction if no carry
        1>D1                     \ set high word of answer to 1
      ;
 1 LIB: =                        \ test for equality
        -                        \ get answer, zero set if equal
        -1>D1                    \ load true
        1D03H Iadd               \ btfss 3,2 skip next instruction if zero set
        0>D1                     \ load false
      ;
 1 LIB: 0=                       \ test if D1=0
        dup 0>D1                 \ make space, set D1=0
        =
      ;
 1 LIB: 0>                       \ test if D1 is positive
        D1>t1                    \ copy D1 to temp1
        0>D1                     \ load false
        1F8EH Iadd               \ btfss 0EH,7 - skip next instruction if negative
        -1>D1                    \ load true
      ;
\ *******************************************************INPUT / OUTPUT WORDS
comment:
The PIC84 has only two eight-bit ports (at addresses 5 and 6), each with a
data direction register (at addresses 85 and 86).  So for I/O we only need
to read and write to four addresses. C5@, C5!, C6@, C6!, C85@, C85!, C86@, C86!
Getting to addresses requires the page-select bits at address 3, bits 5&6, to
be set up.  These are zero on power-up, and we always leave them that way.
comment;
 3 LIB: c5@                      \ byte fetch
        pushD1                   \ make room
        0805H Iadd               \ get value (MOVF 5,0)
      ;

 2 LIB: c5!                      \ byte store
        0085H Iadd               \ store value (MOVWF 5)
        pop                      \ lose value
      ;
 3 LIB: c6@                      \ byte fetch
        pushD1                   \ make room
        0806H Iadd               \ get value (MOVF 6,0)
      ;

 2 LIB: c6!                      \ byte store
        0086H Iadd               \ store value (MOVWF 6)
        pop                      \ lose value
      ;
 2 LIB: c85@                     \ byte fetch
        1585H Iadd               \ BSF 3,5 - point to register block 80
        c5@                      \ read to 5 in that block
        1185H Iadd               \ BCF 3,5 - back to register block 0
      ;

 2 LIB: c85!                     \ byte store
        1585H Iadd               \ BSF 3,5 - point to register block 80
```

(Continues.)

```
       c5!                      \ write from five in that block
       1185H Iadd               \ BCF 3,5 - back to register block 0
      ;
 2 LIB: c86@                    \ byte fetch
       1585H Iadd               \ BSF 3,5 - point to register block 80
       c6@                      \ read to six in that block
       1185H Iadd               \ BCF 3,5 - back to register block 0
      ;
2 LIB: c86!                     \ byte store
       1585H Iadd               \ BSF 3,5 - point to register block 80
       c6!                      \ write to six in that block
       1185H Iadd               \ BCF 3,5 - back to register block 0
      ;
\ *************************************************************PROGRAM JUMPS
: IJUMP ( adr -- )              \ unconditional jump
  2800H [ also forth ] or [ previous ] Iadd  \ lay down jump to adr
;
: IJUMPT ( adr -- )             \ jump if tos true (not 0), consume tos
  pop                           \ flag (tos) to templ
  088EH Iadd                    \ lay down move templ thru alu back to templ (sets
                                \   flags)
  1D03H Iadd                    \ lay down skip next instruction if zero flag set
  Ijump                         \ lay down jump.
;
: IJUMPF ( adr -- )             \ jump if tos false (0), consume tos
  pop                           \ flag (tos) to templ
  088EH Iadd                    \ lay down set flags based on templ
  1903H Iadd                    \ lay down skip next instruction if zero flag clear
  Ijump                         \ lay dowm lump
;

\ *********************************************HANDLING NUMBERS AND VARIABLES
: <INLINE#> ( hi lo -- )        \ lay down code to enter n as an in-line literal
  [ also forth ]                \ need regular nip & and from the FORTH vocabulary
  nip 00ffH and                 \ only using 8-bit numbers for this processor!
  [ previous ]                  \ back to special versions
  load-type @                   \ save previous load type
  2 load-type !                 \ next word being loaded as subsiduary
  dup                           \ push old top of stack
  load-type !                   \ back as we were
  3000H                         \ basic load 0 to D1, overwriting old top-of-stack
  [ also forth ] OR [ previous ] \ convert to load our number to D1
  Iadd                          \ put 8-bit literal in d1
;
: REMOVE#                       \ remove code just laid down to push a #
  [ also forth ]
  I@ 3800H and 2000H =          \ result true if last code was a call
  if -1 else -3 then Iptr +!    \ back up 1 if call, 3 if in-line version
  [ previous ]
;
: VAR-SPACE ( - dadr )          \ allocate space for a variable
  [ also forth ]
  tds @ -1 tds +! 0             \ return as a 32-bit address
  [ previous ]
;
' <Inline#> is Inline#

only forth also notc also definitions
\  Place for processor-specific word to run after the image is generated
: END-ROUTINE
;
```

Forth On-line

It is not our role to interpret the intentions or to verify the claims of resource providers. If you find omissions or errors, send them by e-mail to forl@artopro.mlnet.com. (FORL is an electronic mailbox for tracking publicly available, Forth-related electronic resources; it is provided and maintained by Kenneth O'Heskin.)

Guide to Line Numbers

1.0	Resource name
1.1	Resource startup date
2.0	Location
3.0	On-line address/telephone numbers
4.0	Sponsorship
4.1	Sponsoring person/institution's name
5.0	Contact name (admin, sysop, etc.)
5.1	E-mail address
6.0	Access type (free/pay, conditions of access)
7.0	Connection type (modem/telnet)
7.1	Modem (maximum bps, parity/bits/stop)
7.2	Telnet (address)
8.0	Approximate number of Forth-related files
8.1	Theme of these files
8.2	Available to first-time callers?
9.0	Mail and news
9.1.0	Mail technology
9.1.1	Binary mail tranfers supported?
10.0 ..	System software, if relevant
11.0 ..	Additional comments

Bulletin Board Systems

1.0 Arcane Incantations
1.1 Mar. 93
3.0 617-899-6672
5.0 Gary Chanson
5.1 gary.chanson@channel1.com
8.0 Several files (some authored by sysop), first-time caller available.
10.0 PC Board

1.0 Art of Programming BBS
1.1 Jan. 91
2.0 Mission, BC, Canada
3.0 604-826-9663
4.0 non-profit
4.1 ForthBC Computer Language Society
5.0 Kenneth O'Heskin
5.1 koh@artopro.mlnet.com
6.0 Free dial-up access for all Forth files.
7.0 modem
7.1 v32 8,N,1
8.0 hundreds
8.2 first-time callers ok
9.0 Mail and news; e-mail by low-cost annual subscription; Usenet groups (incl. comp.lang.forth); BCbbs.net.
9.1.0 uucp, qwk
9.1.1 uuencode/decode
10.0 WildcatJGNT_Mail
11.0 Download aop.zip for a list of all files on the board.

1.0 The FROG Pond BBS
1.1 Aug. 89
2.0 Rochester, NY, USA
3.0 716-461-1924
4.0 non-profit
4.1 The FROG Computer Society
5.0 Nick Francesco
5.1 nickf@vivanet.com
6.0 free
7.0 Modem
7.1 14400 8N1
8.0 5
8.1 languages
8.2 yes
9.0 Fidonet and Internet mail available for all users.
9.1.0 qwk,netmail
9.1.1 uuenc/decode
10.0 Remote Access (for now)
11.0 Download FROGPOND.EXE for self-extracting list of all files. All Forth files available to first-time downloaders.

1.0 Gold Country Forth BBS
2.0 CA, USA
3.0 916-652-7117
5.0 Al Mitchell
8.1 Some product support (password required), many free files.
8.2 Okay for first-time callers.

1.0 LMI Forth BBS
1.1 Oct. 84
2.0 Los Angeles, CA, USA
3.0 310-306-3530
4.0 business
4.1 Laboratory Microsystems Inc. (LMI)
5.0 Ray Duncan
5.1 sysop@lmi.la.ca.us
6.0 free
7.0 modem
7.1 1,200 – 28,800 baud, 8/N/1
8.0 hundreds
8.1 Mostly compatible with LMI Forth products, but also some public-domain Forth stuff.
8.2 yes (except for LMI product updates, which require prior registration)
9.0 Supports Internet e-mail and UseNet News
9.1.0 UUCP
10.0 PC Board 15.2
11.0 The LMI Forth BBS is primarily intended for technical support of LMI customers. However, all members of the Forth community are welcome to upload/download files in the public directories, and to use the LMI BBS for Internet e-mail and reading the UseNet comp.lang.forth conference.

1.0 MindLink!
2.0 Vancouver, BC, Canada
3.0 modem: 604-528-3500 (main) 28.8Kbps
Telnet: mindlink.bc.ca
4.0 Business
6.0 Pay; may log on as guest.
7.0 28.8Kbps, Telnet
8.0 75
8.0 Available only to registered users.
11.0 Two Forth file libraries: Sources.Forth and MsDos.Forth.

1.0 RCFB "The Rocky Coast Free Board"
1.1 Oct. 88
2.0 Golden, CO, USA
3.0 303-278-0364
4.0 private
4.1 Jax
5.0 SYSOP
5.1 jax@well.com
6.0 Free, but must register.
7.1 19200, 8-n-1
8.0 300
8.1 Programming tools and productivity
8.2 Must register online, wait 24 hours.
10.0 PC Board since '88, Linux by mid-96.

FTP Sites

1.0 ANS Forth x3j14
3.0 ftp://ftp.uu.net
11.0 This is the ANS Forth archive; electronic versions of dpANS standards document may be obtained by anonymous ftp from:
/vendor/minerva/x3j14/dpans94.zip (Word for Windows, v.2)
/vendor/minerva/x3j14/dpans94.hqx (Word for Macintosh)

Information on this group's mailing, html, and other resources may also be found in these /vendor/minerva/x3j14 directories.

1.0 Asterix Forth archive
2.0 Portugal
3.0 asterix.inescn.pt /pub/forth
4.0 university
4.1 Computer Graphics and CAD group INESC
5.1 paf@porto.inescn.pt

6.0 anonymous ftp
8.0 hundreds
11.0 First internet site of the GEnie Forth archives, built with the assistance of Doug Phillip's FNEAS server. Mirrored on hp.com.

1.0 Brain
3.0 ftp: brain.physics.swin.oz.au
4.0 university
5.1 Tim Hendtlass <tim@brain.physics.swin.oz.au>
11.0 Tim's book "Real Time Forth" is available in /pub/forth or can by accessed by the URLs:
file://brain.physics.swin.oz.au/pub/forth/rtfppc1.zip (LaserJet)
file://brain.physics.swin.oz.au/pub/forth/rtfpps.zip (PostScript)
Recent material on nanocomputers has been added to the site

1.0 Cygnus Support Ftp Service
3.0 ftp://ftp.cygnus.com
http://www.cygnus.com
5.1 info@cygnus.com (?)
11.0 Has a good file list and appears to support some Forth material not available elsewhere on the net.

1.0 Faré's own small FTP site, Forth subsection
1.1 1994
2.0 Paris, France
3.0 ftp://frmap711.mathp7.jussieu.fr/pub/scratch/rideau/
5.0 François-René "Faré" Rideau
5.1 rideau@ens.fr
6.0 free (anonymous FTP)
8.0 Two FORTH systems, my port of eForth to Linux, and Olivier Singla's FROTH.
8.2 yes
10.0 SunOS4.1.3
11.0 This site does not contain much about Forth, but more is welcome if you upload it. I am developing my own system, TUNES, which is remotely Forth-related, and for which I opened this site.

1.0 Hewlett-Packard
3.0 ftp://col.hp.com/mirrors/Forth
6.0 anonymous ftp
11.0 Mirror site for asterix, recommended for North American users when asterix is busy.

1.0 i/tForth-specific stuff
1.1 Sept. 94
2.0 Eindhoven, Brabant, the Netherlands
3.0 ftp iaehv.iaehv.nl, directory pub/users/mhx
4.0 private
4.1 Marcel Hendrix
5.0 Marcel Hendrix
5.1 mhx@iaehv.iaehv.nl
6.0 free, anonymous ftp
8.0 10 – 20
8.1 i/tForth specific files, not ANS enough to put them on taygeta or such. Some very Intel-hardware-specific (networking, audio CD). i/tForth general info, release notes, previews.
11.0 There is a link on taygeta to this directory.

1.0 Microtronix
3.0 ftp.microtronix.com /pub/forth
5.1 Brian Fox <bfox@microtronix.com>, j.fox26@genie.com
11.0 A new site featuring several original (NMI MaxForth) 68HC11 files.

1.0 SimTel
3.0 ftp://ftp.coast.net/SimTel/msdos/forth
5.1 service@coast.NET
11.0 Several Forth files; and Norm Smith's Until revisions are updated here.

1.0 Yerk
3.0 astro.uchicago.edu pub/MAC/Yerk
11.0 Archive for Yerk system, manuals, and info (the OO Forth successor to NEON); Mops also available here.

1.1 July 95
2.0 Ann Arbor, MI, USA
3.0 ftp://williams.physics.lsa.umich.edu/pub/forth
4.0 university
4.1 Particle Theory Group, Physics Department, University of Michigan
5.0 David N. Williams, sysadmin
5.1 David.N.Williams@umich.edu
6.0 free, low traffic, download only
7.0 anonymous ftp
8.0 12–20
8.1 Forth: personal interests of David N. Williams
11.0 This is one directory at an anonymous FTP site devoted mainly to communication between our group and the particle theory community. Forth and symbolic computing (Schoonschip) happen to be an interest of one of our group.

FTP/Web Sites

1.0 Forth Research at Institut fr Computersprachen
2.0 Vienna, Austria
3.0 http://www.complang.tuwien.ac.at/projects/forth.html
ftp://ftp.complang.tuwien.ac.at/pub/projects/forth.html
4.0 university
4.1 Institut fr Computersprachen, TU Wien
5.0 Anton Ertl
5.1 anton@mips.complang.tuwien.ac.at
6.0 free
11.0 There's also some Forth material that is not referenced on the page, in particular:
ftp://ftp.complang.tuwien.ac.at/pub/forth/
http://www.complang.tuwien.ac.at/forth)

1.0 The Mops Page
1.1 Mar. 95
2.0 Philadelphia, PA, USA
3.0 http://www.netaxs.com/~jayfar/mops.html
4.1 private
5.0 Jay Farrell
5.1 jayfar@netaxs.com
6.0 free web/ftp
8.1 The Mops language by Michael Hore. The Mops system, manual, and Doug Hoffman's Selection Framework are directly available from my pub directory. Other files and resources are linked from other sites via the web page.
10.0 My ISP's Unix boxes, which I connect to using a Mac Quadra 605
11.0 Mops 2.6 is Michael Hore's public-domain development system for the Macintosh. With Forth and Smalltalk parentage, Mops has extensive OOP capabilities, including multiple inheritance and a class library supporting the Macintosh interface.

1.0 Ron's Mac and Apple II archive
1.1 June 95
2.0 Milwaukee, WI, USA
3.0 http://kreeft.intmed.mcw.edu/pf.html
4.0 private
4.1 Ron Kneusel
5.0 Ron Kneusel
5.1 rkneusel@post.its.mcw.edu
6.0 free
7.0 ftp and http
8.0 10
8.1 Forth programs I've written for the Mac and Apple II.
8.2 yes
10.0 httpd4Mac-123a and FTPd 2.4
11.0 Types of files: pretty-printer for LaTeX, Forth on a simulated Apple II in Forth, microcomputer simulator/assembler, fractal-drawing program, CGI applications in Forth for MacHTTP.
To be added soon: Web Forms handlers for MacHTTP/WebStar; updated and "improved" Forth for the Apple][e; simple program to show the period-doubling route to chaos.
Mac files are BinHexed CompactPro archives (transfer as text); Apple][files are ShrinkIt archives (.shk, binary).

1.0 taygeta.oc.nps.navy.mil
1.1 1990
2.0 Monterey, CA, USA
3.0 taygeta.oc.nps.navy.mil (131.120.60.20)
www: http://taygeta.oc.nps.navy.mil/fig_home.html
4.0 non-profit
4.1 Skip Carter
5.1 skip@taygeta.oc.nps.navy.mil
11.0 A premiere Forth archive on the net; includes Forth Scientific Library, CD-ROM project, GEnie archives.

1.0 University of Bremen
3.0 //ftp.uni-bremen.de/pub/languages/programming/forth http://ftp.uni-bremen.de/FTP/ftp.html

5.1 ftp-admin@ftp.uni-bremen.de
11.0 Features a full ../Taygeta-Mirror archive (information from c.l.f post by dku@zarniwoop.cp-labor.uni-bremen.de (Dirk Kutscher).

Internet Mailing Lists

1.0 ANSForth Mail Group
3.0 ANSForth@minerva.com
(mail group membership)
ansforth-request@minerva.com
(to join ANSForth mail group)
11.0 Communicate with these for questions pertinent to the Technical Committee X3J14 working on ANS Forth.

1.0 FIRE-L
1.1 Sept. 94
2.0 global
3.0 subscribe:
listserv@artopro.mlnet.com
submissions:
fire-l@artopro.mlnet.com
5.0 Moderated by Rick Hohensee
5.1 rickh@cap.gwu.edu
11.0 The Fire-l Mailing List is for updates, discussions, debate, speculation, and announcements of Rick Hohensee's free-form FIRE specification.

1.0 LMI Technical Support
3.0 support@lmi.la.ca.us
4.1 Laboratory Microsystems Inc.
5.1 Ray Duncan, e-mail:
<ray.duncan@lmi.la.ca.us>
11.0 Technical support for users of LMI Forth systems.

1.0 MISC mailing list
3.0 Subscribe to:
misc-request@pisa.rockefeller.edu
Articles: misc@pisa.rockefeller.edu
5.0 Jeff Fox and Penio Penev
5.1 jfox@netcom.com (Jeff Fox)
Penev@venezia.rockefeller.edu (Penio Penev)
11.0 The MISC list is about all aspects of the new P21/P8/P32 and F21 Minimal Instruction Set technologies being developed by Charles Moore and his MISC associates.

1.0 The Win32For mailing list
1.1 Dec. 94
3.0 for list entries:
win32for@edmail.spc.uchicago.edu
for un/subscribe (on one line):
win32for-requests@edmail.spc.uchicago.edu
5.0 Carl Zmola
5.1 zmola@cicero.spc.uchicago.edu
11.0 Discussion of all Win32For issues, the Win NT/95 object-oriented Forth system from Andrew McKewan and Tom Zimmer.

Electronic Mailboxes

1.0 The Forth Online Resources Survey (FORL)
1.1 July 95
3.0 forl@artopro.mlnet.com
11.0 A permanent mailbox/index for tracking the ebb and flow of all publicly available Forth electronic resources.

1.0 Miller Microcomputer Services
1.1 Dec. 90
2.0 Natick, MA, USA
3.0 dmiller@im.lcs.mit.edu
4.0 business
4.1 Miller Microcomputer Services
5.0 A. Richard Miller
5.1 dmiller@im.lcs.mit.edu
6.0 free
7.0 Internet
8.0 none
9.0 none
11.0 We stock Forth-related books (some on sale) and MMSFORTH software. We support licensed users of MMSFORTH, FORTHCOM, FORTH-WRITE, DATAHANDLER-PLUS for IBM-PC (MS-DOS and non-DOS/standalone). We provide PC-compatible consulting and hardware. Request our free e-mail brochure "MMSFORTH and Forth books."

Newsgroups, Conferences, et al.

1.0 comp.lang.forth
11.0 Usenet newsgroup c.l.f is the premiere global Forth bulletin board. Articles from comp.lang.forth are archived at: ftp://asterix.inescn.pt/pub/forth/news/

1.0 GEnie
11.0 GEnie is run by General Electric Information Services (GEIS). Its Forth "RoundTable" has a bulletin board and library. For info, including local access numbers (not just U.S. and Canada), phone 800-638-9636. "As a user and worker on GEnie, I have found customer service to be very good."

World-Wide Web

1.0 Bernd Paysan's Web site
3.0 http://www.informatik.tu-muenchen.de/cgi-bin/nph-gateway/hphalle2/~paysan/
5.0 Bernd Paysan
5.1 paysan@informatik.tu-muenchen.de
11.0 BigForth and GForth systems and information.

1.0 FIG home page
3.0 http://www.taygeta.com/fig.html
4.0 non-profit
4.1 Forth Interest Group
5.1 e-mail: johnhall@aol.com
11.0 The FIG home page includes an on-line membership form, a comments mailer, and links to several other Forth URLs—e.g., Andrew Bartelt home page
http://www-ece.rice.edu/~andrew
M. Borasky's Home Page
http://www.teleport.com/~znmebp7

1.0 FORTH, Inc. Home Page
1.1 June 95
2.0 Los Angeles, CA, USA
3.0 http://www.earthlink.net/~forth
4.0 business
4.1 FORTH, Inc.
5.0 E. Rather
5.1 erather@forth.com
6.0 free website
11.0 Site includes summary info and detailed data sheets for FORTH, Inc. products, Forth programming course outlines, application descriptions (some with photos), and links to other Forth sites. Material added periodically.

1.0 F-PC Homepage
1.1 May 7, 1995
3.0 http://www.efn.org/~fwarren/fpc.html
5.1 Fred Warren, email:
fwarren@gears.efn.org
8.1 Related to F-PC Forth for the IBM-PC.
11.0 Homepage dedicated to the Forth for the IBM-PC known as F-PC. It is a full-featured, non-ANSI-compliant (superset of Forth-83 standard) public-domain version of Forth. This page introduces Forth and F-PC, can download F-PC and tutorial material, and on-line mini-tutorials about features of F-PC. This page will eventually be a repository for useful F-PC libraries.

1.0 Jeff Fox's Home Page
1.1 Dec. 93
2.0 Berkeley, CA, USA
3.0 http://www.dnai.com/~jfox
4.0 Business
4.1 Ultra Technology
5.0 Jeff Fox
5.1 jfox@netcom.com (most often)
jfox@dnai.com (supports Eudora)
8.0 40 files
8.1 UltraTechnology, Computer Cowboys, Offete Enterprises, MISC chips, P8, P21, F21, P32, parallel programming in Forth, and AI.
9.1.1 uuenc/decode (on the netcom account)
11.0 This web site is organized by subject from the home page listed above. Incl. individual home pages for my company, Ultra Technology (http://www.dnai.com:80/~jfox/ultra.html); Chuck Moore's company (cowboys.html); Dr. Ting's company (offete.html); and for Minimal Instruction Set Computers (misc.html); as well as for MISC chips like P8, P21, and my chip, the F21.
There are FORML Conference papers, and *FD* articles in html format. There is a copy of the first published article on Forth by Chuck Moore in 1970 (4th_1970.html). Many documents are available in html, .DOC, .ZIP, .PRN, .TXT, with some .EXE, etc. All files are cross-indexed in ultrafre.html, which is listed as "Free Files" on my home page.

(Continues on page 37.)

Stretching Forth

Associative Lists

Wil Baden
Costa Mesa, California

Associative Lists as Wordlists

This article is in response to an inquiry about how to implement associative arrays.

On an external medium, particularly disk, a *database* is a collection of *keys* with associated *properties*. Keys may or may not be able to be inserted or deleted; properties may or may not be able to be modified.

In direct access memory, such an object is a *symbol table*, or *associative array*, or *associative list*.

Every programming language processor has such a mechanism. Forth is rare, if not unique, in that the compiler's own facility is available to the programmer.

When challenged to implement associative arrays, the Forthly response is: Use `WORDLIST` from the Search-Order wordset.

Let's see how we could use it to define a word to define associative-lists.

```
: associative-list WORDLIST CONSTANT ;
```

Well, that was easy. Now let's define some words to use them.

Just three other words from the Search–Order wordset, SET-CURRENT, DEFINITIONS, and SEARCH-WORDLIST, give us what we need for basic operations with associative-lists. (See Figure One.)

See Listing One [page 38] for the relevant words from the Search-Order wordset. See the appendix [page 38] for string operators S+ and S, used here.

Examples

Let's begin setting up two associative lists, Internet Country Codes, and U.S. Postal Service State Codes. That all our entries are two letters long is not significant. (See Figure Two.)

Associative Lists from Scratch

Wordlists as vocabularies and wordlists as associative arrays are used differently. A member of a wordlist as vocabulary is translated and executed automatically when encountered in the input stream with an appropriate search-order enabled. A wordlist as associative array generally takes at least three explicit specifiers to get an associated property: a value for the key, an identifier for the associative array, a function for the property.

In a wordlist, the keys must follow the rules for Forth words in the implementation, and there is a stringent limit as to the number of wordlists that can be defined. To overcome these limitations, you can define associative lists directly. The look-up method that will be used here is brute force, which should always be your first choice.

This implementation is about twice as big as the wordlist approach, but is still economical. (See Figure Three.)

The examples are the same as above. Applications and extensions are left to the reader.

Conclusion

Being able to do-it-yourself is the good news; having to do-it-yourself is the bad news. Doing-it-yourself is more fun.

(Listing One and Appendix appear on page 38.)

Wil Baden is a professional programmer with an interest in Forth. wilbaden@netcom.com

Figure One. Definitions for basic associative-list operations.

```
: entry                                    ( s . wid -- )
     SET-CURRENT                           ( s .)
          S" CREATE " 2SWAP S+ EVALUATE    ( )
     DEFINITIONS
;

: item?                                    ( s . wid -- 0 | xt )
     SEARCH-WORDLIST DUP IF DROP THEN
;

: item                                     ( s . wid -- a )
     item? DUP 0= ABORT" Not an item. "    ( xt)
     >BODY                                 ( a)
;

: items                                    ( -- )
     SET-CURRENT
          WORDS
     DEFINITIONS
;
```

Figure Two. Setting up two example associative lists.

```
associative-list country
associative-list usa

( Define three country codes and two state codes. )

S" uk" country entry S" United Kingdom" S,
S" ca" country entry S" Canada" S,
S" ca" usa entry     S" California" S,
S" de" usa entry     S" Delaware" S,
S" de" country entry S" Germany" S,

( Here's how to get the property of a code. )

S" ca" country item COUNT TYPE     \ Canada
S" ca" usa item COUNT TYPE         \ California

( List the country codes defined so far. )

country items                      \ de ca uk
```

Figure Three. Implementing associative lists from scratch.

```
: associative-list CREATE 0 , ;

: entry                           ( s . list -- )
     ALIGN    HERE SWAP                ( . . here list)
     DUP @ ,
     !                                 ( s .)
     S, ( ) ALIGN                      ( )
;

: item?                           ( s . list -- a )
     @                                 ( s . pointer)
     BEGIN
          DUP
     WHILE
          3DUP CELL+ COUNT  COMPARE
     WHILE
          @
     REPEAT
          CELL+ COUNT CHARS  +  ALIGNED ( s . a)
     THEN
     NIP NIP                           ( a)
;

: item                            ( s . list -- a )
     item? DUP 0= ABORT" Not an item. "  ( a)
;

: items                           ( list -- )
     @                                 ( pointer)
     BEGIN
          DUP
     WHILE
          DUP CELL+ COUNT  TYPE    SPACE
          @
     REPEAT                            DROP
;
```

("Forth On-line," from page 35.)

1.0 Leo Brodie Services
3.0 http://www.pacificrim.net/~lbrodie/lbs.html
5.0 Leo Brodie
11.0 Renowned author of *Starting Forth* and *Thinking Forth* keeping in touch with the community.

1.0 Nick Francesco's Forth Page
1.1 Feb. 95
2.0 Rochester, NY, USA
3.0 http://raptor.rit.edu/Nick/forth.htm
4.0 Private
4.1 Nick Francesco
5.0 Nick Francesco
5.1 nick@rit.edu
6.0 free
7.0 Web Browser
8.0 5
8.1 Forth resources on the net
8.2 yes
9.0 none
11.0 The Sound Bytes Radio Show Home Page: http://www.vivanet.com/soundbytes

1.0 Open FirmWare
3.0 http://www.firmworks.com
4.1 Bradley Forthware
11.0 Several Postscript documents available from this URL on PowerPC, Sparc for Open FirmWare development, licenses, and consultation.

1.0 Phil Koopman's Forth Mini-Page
1.1 July 95
2.0 East Hartford, CT, USA
3.0 http://danville.res.utc.com/Mechatronics/ads/koopman/forth/index.html
4.0 personal
5.0 Philip Koopman
5.1 koopman@utrc.utc.com
6.0 free
8.0 Personal Forth and stack machine publications
11.0 In html as of July 1995:
- WISC CPU/16 patent cover page and block diagram.
- WISC CPU/32 (Harris RTX-4000) patent cover page and block diagram.
- Preliminary exploration of optimized stack code generation (*JFAR* paper).
- Brief introduction to Forth ("two-page" language overview).

1.0 Pocket Forth Home Page
1.1 June 95
2.0 Phoenix, AZ, USA
3.0 http://chemlab.pc.maricopa.edu/pocket.html
4.0 Private on a community-college-owned computer.
4.1 Chris Heilman/Phoenix College
5.0 Chris Heilman
5.1 heilman@pc.maricopa.edu
6.0 free/daytime access may be slow or limited

(Continues on next page.)

Listing One. Relevant words from the Search-Order wordset.

16.6.1.1180 **DEFINITIONS** SEARCH

(--)

Make the compilation word list the same as the first word list in the search order. Specifies that the names of subsequent definitions will be placed in the compilation word list. Subsequent changes in the search order will not affect the compilation word list.

16.6.1.2192 **SEARCH-WORDLIST** SEARCH

(*c-addr u wid* -- 0 | *xt* 1 | *xt* -1)

Find the definition identified by the string *c-addr u* in the word list identified by *wid*. If the definition is not found, return zero. If the definition is found, return its execution token *xt* and one (1) if the definition is immediate, minus-one (-1) otherwise.

16.6.1.2195 **SET-CURRENT** SEARCH

(*wid* --)

Set the compilation word list to the word list identified by *wid*.

16.6.1.2460 **WORDLIST** SEARCH

(-- *wid*)

Create a new empty word list, returning its word list identifier *wid*. The new word list may be returned from a pool of preallocated word lists or may be dynamically allocated in data space. A system shall allow the creation of at least 8 new word lists in addition to any provided as part of the system.

Appendix. Definitions of string operators s+ and s,.

```
: S,                                ( s . -- )
     DUP C,
     0 ?DO                              ( s)
          COUNT C,
     LOOP                               DROP
;

: S+                                ( s1 . s2 . -- s3 . )
      >R                                ( s1 . s2)
           OVER  CHARS PAD +  R@         ( s1 .   s2 a .)
          MOVE                          ( s1 .)
           >R                            ( s1)
              PAD R@ MOVE               ( )
               PAD                      ( s3)
      R> R> +                           ( s3 .)
;

: 3DUP                            ( a b c -- a b c a b c )
     2 PICK  2 PICK  2 PICK             ( a b c a b c)
     ( or DUP 2OVER ROT )
;
```

("Forth On-line," from preceding page.)

7.0 www only.
8.0 about 40
8.1 Pocket Forth
8.2 yes
9.0 Click a link to e-mail the author of Pocket Forth.
10.0 Mac OS
11.0 This site is maintained by the author of Pocket Forth and includes archives of software written in Pocket Forth, such as programming demos, applications, and unique CGI programs written in Pocket Forth.

1.0 Stephan J Bevan's Web page
3.0 http://panther.cs.man.ac.uk/~bevan/forth
5.1 bevan@cs.man.ac.uk (Stephan J. Bevan)
11.0 Up-to-date FAQ information on Forth implementations and books; e-mail maintainer to make suggestions, corrections, and additions.

1.0 The Computer Journal
3.0 http://www2.psyber.com/~tcj/groups.html
5.0 TCJ Editor: Bill D. Kibler
5.1 tcj@psyber.com
11.0 "Hands on support for the Trailing Edge of Technology" is the tongue-in-cheek motto of this publication about old and new hardware and embedded systems programming; URL has many pages of info, much with direct bearing on Forth.

1.0 The TUNES project
1.1 1995
2.0 Paris, France
3.0 http://www.eleves.ens.fr:8080/home/rideau/Tunes/
5.0 François-René "Faré" Rideau
5.1 rideau@ens.fr
6.0 free (GNU copyleft)
8.0 Only part of one file points to Forth www sites, but the Forth spirit has contaminated the whole project.
8.1 Review of actual Forth in .../Review/Languages.html#FORTH and of my own version of Forth in .../LLL/LLL.html.
8.2 yes
10.0 SunOS 4.1.3
11.0 This site is for my TUNES system project, only remotely related to Forth. The only thing about actual Forth is: http://www.eleves.ens.fr:8080/home/rideau/Tunes/Review/Languages.html#FORTH

(Fast Forthward, continued from page 43.)
C++ programmers. (Forth programmers have the choice of using stack comments to help give descriptive labels to dynamic data.)

Named dynamic data can make program source code easier to comprehend. This important facility involves the widely used feature of local (automatic) variables in languages like C. This continues to be the practice because local (automatic) objects are featured in C++.

When pointer names are used to refer to dynamic data (local objects or variables), the pointer name is usually appropriately descriptive, too.

For C++, the typed pointers of C have been embellished with all the extra semantics and syntax of C++ objects when the thing they point to is an object. So if `operand1Ptr` is a pointer to a complex number object,

```
operand1Ptr->add(operand2)
```

should reference the member function for the addition of two complex numbers.

However, two object instantiation protocols are required in OOLs like C++. A new operator creates dynamic objects in C++ at the runtime for a routine. A data-type (class) name can also be used to create object instances as part of the initialization of static data. It can also be used to create compiler-scoped names for pointers to the associated class of object.

In the pointer case, no memory is actually allocated. For pointers, memory allocation is not an automatically supplied step, but one that you code explicitly. Thus, there are two different ways to bring a dynamic object into existence in C++:

- If the data-type name is employed inside a function for the creation of an object, a named local (dynamic) object is created with allocated memory. It takes the same object syntax as would normal static objects.
- If an object is instantiated inside a function using the new operator, then a pointer-style of syntax applies to the address that is returned by new. Usually, the returned address is assigned to a type-corresponding pointer that was declared to the compiler earlier, as in:

```
complex*operand1Ptr;
...
operand1Ptr = new complex;
```

Soft-coding Functions and Data Types

One reason we should wish to encounter objects *along the way to* functions is that we can go soft on the functions that need to be called. For example, if I want an addition suitable for double number arguments, I can ask for *addition* as it applies to an operand of the double data type, as follows:

```
operand1.add(operand2)
```

The compiler can determine the exact form of addition needed.

Because a class aggregates the data types from the line of parent classes from which it is inheriting, what I have been describing as polymorphism is perhaps better described as inheritance. In any case, member functions from one or more parent classes will be mated to objects of the data type corresponding to their subclass.

Soft-coded functions and data types is the distinction of objects that I find most remarkable, regardless of the label we put on it. Accordingly, I'll continue to attach the label of polymorphism to continue to emphasize that the member function calls for objects are class-encapsulated and, thereby, soft-coded.

Note how little maintenance is required if we change the declaration of `operand1` and `operand2` to make them floating-point numbers. The OO compiler merely gives me an addition operation suitable for floating-point numbers. I don't have to weed through my code, unearthing all the obsolete references to a uniquely named addition operation for double numbers.

Of course, C, BASIC, and Pascal already had this level of support in terms of functions that were exposed through operation symbols. However, this was only the case for built-in data types. A modern OOL extends the convenience enjoyed with soft-coded operations to soft-coded functions.

To be able to treat the programmer-supplied functions as flexibly (polymorphically) as built-in (overloaded) operations, data types must be declared that overload member functions with common names (or common operation symbols). The decision to create classes with analogous member functions but different data structures is wise, even though it can be more work, because of the malleability it can afford.

Through the practice of overloading operator symbols or member function names in several closely related classes, we can anticipate the need for data types to change once a program is complete. Using an alternate data type to represent certain values can dramatically alter the performance, memory-appetite, and other run-time characteristics of the final program.

I think of this feature as a sort of what-if scenario-exercising facility for data-structuring decisions you must make. It is a playful feature, in the same sense that Forth's interactive style of programming can be so immediate that it becomes a pleasure to use.

The data-structuring decisions you must make early in the development cycle for an application can remain in flux until the end of development. Even after development has ceased, the overall characteristics of your program are subject to a certain amount of fine-tuning with small changes to the source code.

Of course, this potential benefit is only realized if we can program or purchase classes or class libraries that can readily be substituted for one another. C++ has recently adopted a template library that has code-generation capabilities for ensuring just such an outcome for the more commonly required classes, such as queues. For example, using the template library, queue classes can be readily constructed for integers, floats, and even any programmer-supplied data types.

Arguably, it may be wise to create classes with member functions for which there are no analog member functions inside analogous classes. This merely forces an object form of a function-calling syntax upon our code, making functions soft-coded. Later, we are free to add the look-alike classes so that the flexibility of soft-coded functions can be enjoyed. If nothing else, we can use this facility in place of conditional compiler directives.

The Matchmaking of Casts and Objects

Automatic casts are a close cousin to polymorphism. Through automatic casts, function parameters can be coerced to match the data type expected by the function. This avoids data-type mismatches, too, providing that the conversion operation applied can be considered robust.

Automatic casting allows the effective function to remain the one explicitly specified. In contrast, polymorphism selects a function, if any, that suits the unchanged data type of the related object.

For example, polymorphic selection of an unsigned addition operation due to the presence of an unsigned integer operand is more robust than for the compiler to automatically cast the unsigned number to a signed number for use with a signed-integer addition.

Such a cast would involve a loss of one bit of resolution. Automatic casting can be considered completely safe when the type is made more general. Unsigned numbers do not possess an "is-a" relationship with signed numbers. In fact, neither a relationship of signed-is-an-unsigned-number or a relationship of unsigned-is-a-signed-number holds true.

Considering how data types inherit from their parent classes, every node data type is considered to be an aggregation of the data types from its parent classes. So any cast of an object to an object of a more general data type is effortless and safe. No conversion operation need be compiled to change the data type of the object.

By design, mating member functions will exist for it, perhaps drawn from a parent class. Through early binding, this involves no run-time overhead, nor even a function pointer.

(The same claim about type safeness is not true when down-casting to a more specific data type. For safe handling in such a situation, the requirements are virtual functions in abstract classes and pointer-moderated function calls for member functions at runtime, also known as late binding.)

Due to class-structuring rules, neither signed nor unsigned numbers can be the parent class of the other. As a result, automatic casting between them can be considered suspect because a conversion function is required. Depending on the conversion function, the converted data may be unfaithful to the original.

(Recall that C++ is a hybrid language, however, so it can break its own object-oriented rules.)

C was much less type safe. C++ is, nevertheless, free to continue with the old ways of C, which it sometimes does for the sake of backwards compatibility.

Unfortunately, C++ also permits casting between data types that are not part of a class inheritance chain. Where is the wisdom in being able to convert an integer to an array? By adding this additional form of automatic casting, the type-checking facilities of C++ are considerably weakened. (A workaround has been approved, however, as part of the continuing evolution of the C++ standard.)

The Definitiveness of Data

An increment function for a variable used for sample-counting is profoundly different than an increment function for a variable used to hold a temperature threshold. We do not understand the significance of a function action until we understand the nature of the data it acts upon.

Since many representations are possible for the same data, its data type should be considered a subordinate attribute. For example, if a sample-counting variable is changed from an integer to a double, its algorithmic role remains unaffected.

A shift to a data-centric view is therefore justified to emphasize the data apart from its mutable type—and, by extension, its mutable type-associated operations. It's as though we need to tell the compiler: "When incrementing the value for temperature threshold, use its associated data type to resolve the exact function for that type, regardless of what that data type may be today, tomorrow, or the next day."

Through polymorphism, objects offer the encapsulation tool with which to soft-code functions and data-types, affording more maintainable programs. In contrast, a function-centric way of coding of program actions runs a risk of obsolescence due to a greater likelihood for data-type changes. Even when the data types of function parameters are built-in types, automatic casting imparts only limited flexibility, beyond which code obsolescence is produced, and maintenance work is engendered.

(Functions with indeterminate types of parameters are exceptional, such as `printf()`. For such functions, type-checking is unavailable because data type mismatches are not caught early.)

An Object-led Programming Style

A style guideline I constantly hear from gurus in both the Forth and C encampments is to eliminate literal numbers from your programs and replace them with enumerations or named constants.

I claim that the practices of data-type encapsulation and literal-value encapsulation are similar because both practices produce a program that is easier to maintain and fine tune. As an example of literal-value encapsulation, a program previously written to assume a line length of 80 columns can be rewritten in terms of a line-length constant. For a program that encapsulates the line-length in a constant, very trivial changes can support alternative line widths. Likewise, by changing the data declaration from a constant to a variable, and making a few more simple changes, run-time changes of the line width can be supported.

You can also understand these forms of encapsulation in terms of the use of a client-server protocol. Client functions throughout the program that require knowledge of the line width are served that value through the variable or constant that encapsulates it.

With OOLs, the encapsulation capacity is taken up a level: the sites of function calls serve up the correct functions. These calling sites are soft-coded references to functions. Function selection relies upon the data type of a mandatory accompanying reference to an object instance. The object name and the function name thus work together to identify the correct function.

These two types of encapsulation have a similar look and feel. For example, several print functions may be written for use in conjunction with various types of line-width objects. Auxiliary line-width data types might be created to hold a length as well as an algorithm that permits a right-adjusted line-print style, or an algorithm that centers lines.

When the `LINE_WIDTH` instance object is re-declared using an alternative (compatible) data type, an entirely new function can be resolved at the site of a soft-coded function reference such as:

```
LINE_WIDTH.print(line)
```

If this sounds a bit like the tail wagging the dog, well so be it.

Just as the gurus permit no exceptions to the elimination of literal values from programs, I suspect OO programming shines its brightest when there are no exceptions to the soft-coding of functions and data types. Beware, however, because this type of issue leads to the debates that continue between the object purists and object nonpurists.

We need to answer the question: Is it ever beneficial to hard-code functions and data types? (I don't believe it is ever beneficial to literally specify a value as opposed to encapsulating it in a constant.)

Another guideline for an object style of programming is the pursuit of namespace reduction. It is natural to want to view a proliferation of functions as an order-of-magnitude fewer functions, if at all possible. One way we can make hordes of functions more manageable is by naming related functions the same, but placing them in different classes. That way, a more limited number of function names can make less direct (i.e., soft) reference to a larger pool of related member functions, according to object contexts.

However, this argument appeals most when you are coding larger applications. Programming guidelines are largely self defeating when they vary in applicability according to circumstances, such as program length.

Because namespace reduction is a natural byproduct of the guideline to soft-code functions and data types using polymorphism, it need not be considered a separate and distinct guideline.

Conclusion

Even if you have despised object-oriented programming before now, you should be able to see that soft-coding permits the final application to be more easily maintained and fine tuned. This is in the same spirit as Forth's incremental development.

Objects may be seen as a part of a larger pursuit of more mutable or fine-tunable programs. If there exists a means to achieve a similar goal that can bypass the conventions and peculiarities of OOLs, then we could perhaps afford to ignore object languages.

We Forthers may be critical of object-oriented programming languages for various and compelling reasons. However, the goal of soft-coding function calls through the data type of an accompanying object reference remains a laudable one.

Considering how operator polymorphism was already a part of modern programming languages, OOLs are shown to be directly descendent of previous languages. By emphasizing those ties to previous languages, I have shown that OOLs do not have to be considered foreign. A germ of their goals and their mechanisms for implementation, such as operator selection based on data types, already existed. Automatic casting is another prevalent facility that is closely related to objects.

In any case, we should not dismiss OOLs without a serious study of them. If they do not determine the future of programming, objects at least signal a shift in programming styles. Objects will make programming easier insofar as they support flexible programming without bringing us programmers too much additional grief and without making our programs inefficient.

C++ does well in terms of not making our programs inefficient. Its ability to offer the efficiency of hard-coded C functions, but the convenience of soft-coded functions, is a remarkable achievement.

As an OOL compiler, C++ helps eliminate operator and operand mismatches—even for programmer-supplied data types and for functions and their parameters—primarily through its better exploitation of data-type information encapsulated by references to data objects of specific classes.

However, I suspect C++ is causing a lot of unnecessary grief.

C++ places upon us the additional burdens of an unwieldy language syntax and compilers that are both too smart and too dumb. To implement a data type thoroughly, we are asked to enter into the delicate business of writing constructors and destructors, the likes of which were never exposed for the built-in data types of earlier programming languages. It's as if we are being asked to take over some of the compiler-writing role. In this regard, the C++ compilers are too dumb.

The compilers are also too smart, because they cling to the role of writing constructors and destructors for you. In many cases, the work these smart compilers would do for you is usually useless due to oversimplification. So we need to be able to take control from them, meaning that we must out-smart these smart compilers. In this regard, these compilers are too smart.

Forth supports no soft-coding of functions that is moderated by data types. Change a Forth integer variable to a double, and you must go hunting for all references to integer-specific operations with that variable.

Of course, admirers of OOLs will see Forth as an elegant candidate for building an OOL, because its present status makes backward compatibility irrelevant. It has the opportunity to support objects, and built-in and other data types, using a common protocol. This should be much better than kludging together two separate mechanisms, in the manner of hybrid languages like C++.

Objects Promote New Programming Style

Mike Elola
San Jose, California

One cannot conceive of programming without functions. Functions correspond to program actions. Each program action corresponds to one or more groups of instructions that comprise functions. Because functions can easily refer to one another, they tend to become a meta-language for specifying programs.

With our long labors directed towards defining the best functions for specifying an application, it is natural for us to look upon data structures as lesser concerns. Oddly, however, there are good reasons to elevate our consideration of data structures to an equal level, or even a level above functions.

Even if we pack the functions into modules, the names and calling conventions of the functions always remain the focus of our reuse effort. Likewise, if we package functions inside object classes, we still continue to define the actions of a program in terms of class-member functions, if not normal functions. Part of the calling convention for a member function is a mandatory reference to an instance object of a related class.

Objects can bring a better style of programming while easing code maintenance.

The name of an archive library or DLL can be more or less arbitrary. Changing the name of a library does not affect the source code that implements program actions. In just a few places, coordinated changes are required. For example, compiler invocation commands will be affected.

Because we deploy functions at widely varied locations throughout our source code to specify program actions, changes to function names or to the parameters they accept can make many passages obsolete throughout our source files.

Although macro processors (such as the C preprocessor) allow the soft-coding of functions so that less maintenance arises from such changes, objects can bring a better style of programming while easing code maintenance.

Data is not normally referenced without the intent of processing it somehow. So the suffixing of an object reference with a function reference is not only natural, it is a clever extension of the normal function-calling syntax. Considering how the function reference alone can be ambiguous, the prepended object reference supports its timely disambiguation.

For this reason, using object-moderated functions can help us more than trafficking in normal functions alone.

Before expanding upon this idea (in "Soft-coding Functions and Data Types"), the expanded role of data types will be explored in three or four brief sections following. C++ will be used as a reference language for comparing object programming to conventional programming.

It can also be illuminating to compare objects to variables. For variables, overloaded operators had to be matched to appropriate functions, such as plus and minus. For objects, overloaded member functions may need to be discriminated between.

Like ordinary variables, objects are identified with a single data type. Unlike ordinary complex data types (C structs or Pascal records), however, objects are tapped for their associated data-type (class) information by a C++ compiler to resolve overloaded names or symbols to type-suitable functions.

Variables come in two flavors in languages such as C and C++. There are static globals (described in "Static Variables") and there are dynamic or automatic variables (described in "Dynamic Variables and Data").

Extensibility Extended to Data Types

A primary feature of object-oriented languages (OOLs) is the extension of data typing in such a way that programmer-supplied data types can be indistinguishable from built-in data types (a Forth-inspired trend, no doubt).

We Forthers already have a deep appreciation of a programming language that offers transparent extensibility (an acquired taste?). Perhaps we will discover a greater kinship with our C++ brethren as they acquire a similar appreciation.

Enlarged Data Type Scope

Objects bring a new form of extensibility to data-typed programming languages. We won't fully appreciate this until we first appreciate how OOLs redefine what constitutes a (complete) data type.

Before objects, the notion of a data type had been defined very narrowly. With part of their implementation unexposed (operator functions and automatic casts) and partially exposed (data declaration syntaxes and manual casts), data types were largely off-limits to programmers.

What had gone unrecognized, and what objects now expose, is that data types can be broadened to account for such closely related provisions as operator overloading, member function overloading, and polymorphism. As an example of operator overloading, the plus sign can usually refer to floating point as well as integer addition, depending on context.

(Forth is unusual with respect to its neglect for operator overloading; it merely offers a larger number of type-specific primitive operations.)

The OOLs attempt to carry the data-type-related feature of polymorphism out of the backwaters of built-in data types, extending it to programmer-supplied data types and their associated (member) functions with overloaded names. Data types have been enlarged by OOLs into a module-like conglomerate that includes function declarations as well as data-structure definitions.

This may sound like smoke and mirrors. Some will say that it is a simple matter of better code packaging, since classes merely consolidate every declaration pertinent to a data type into one easily identified unit of code.

Data Types Annex Functions

Because it is redefined as a broader unit of code packaging, a data type (or class) is a more flexible container for holding elements of the source code for an application.

However, saying that objects merely better organize code overlooks their expanded role. For one thing, once the conglomeration of related provisions constituting a data type is bundled inside a well-delimited lexical unit with a unique name, the collection as a whole can be easily referenced elsewhere.

This makes the redeployment of data structures and associated function declarations easy. In such a way, one data type can easily be nested inside of another data-type (or class), without a polluting it and without any code duplication.

Data-type Factoring Revealed

Objects support a form of data-type factoring that is moderated through classes.

When data types are properly formalized as classes, an identical process to what we Forthers call factoring becomes possible with respect to the syntactic unit formerly known as a data type (now known as a class). Built-in and other previously defined data types can be aggregated to produce new data types (new classes).

Static Variables

Static or global data structures are given names that benefit our comprehension of source code. They also disclose various program states. These same benefits and more will be enjoyed for static object instances as well.

(Persistent objects can also be considered static objects. Any data location that is not allocated and freed with the lifetime of a program is static. Persistence usually implies preservation of data—not only during the lifetime of the program, but across runs of the same program, and even across runs of different programs that share data. Accordingly, the persistence of Smalltalk objects refers to a longer-lived object than a C++ static object.)

Another way we can appreciate static data is as a key to understanding a program written by someone else. Because they are often assigned values that reflect program states, static variables can be important landmarks. (Variable names help document the program better. If names are chosen with care, this extra documentation makes any program much easier to understand.)

While the dynamic data on the stack can reveal a dizzying amount of processing, the slower-motion activity going on with respect to the variables can offer us a "big picture." With appropriate tools, we can monitor variables at run time to help us divine the overall direction of the finer-grained processing.

When state information is evaluated in if-statement predicates, the state information helps determine appropriate processing, such as whether to compile an in-line double or an in-line integer. Viewing the Forth interpreter and compiler as state machines, certain Forth system variables capture the state of the system:

- `STATE`—captures whether the next word in the input stream should be treated as interpretation source or compilation source.
- `DPL`—captures whether a number parsed from the input stream was a double or an integer.
- `CURRENT`—points to the latest word in the current vocabulary.
- `CONTEXT`—points to the latest word in the context vocabulary.

Some will say it is a matter of better code packaging.

- `DP`—points to the next free compilation address.
- `BLK`—points to the block (or file handle).
- `IN`—points to the current character position inside the input buffer.

These static variables help preserve data that should not be lost between calls to `LOAD`, `INTERPRET`, and other entry points for the Forth input stream processor.

Dynamic Variables and Data

Dynamic data has a shortened lifespan relative to the length of a program run. The memory used by dynamic data is subject to recovery and reuse.

Names may be omitted for dynamic data on the stack or on the heap, although this is rarely the practice of C and

(Continues on page 39.)

CALL FOR PAPERS FORML CONFERENCE

The original technical conference for professional Forth programmers and users.

Seventeenth annual FORML Forth Modification Laboratory Conference
Following Thanksgiving November 24–26, 1995

Asilomar Conference Center
Monterey Peninsula overlooking the Pacific Ocean
Pacific Grove, California USA

Theme: Forth as a Tool for Scientific Applications

Papers are invited that address relevant issues in the development and use of Forth in scientific applications, processing, and analysis. Additionally, papers describing successful Forth project case histories are of particular interest. Papers about other Forth topics are also welcome.

Mail abstract(s) of approximately 100 words by October 1, 1995 to FORML, PO Box 2154, Oakland, CA 94621. Completed papers are due November 1, 1995.

The Asilomar Conference Center combines excellent meeting and comfortable living accommodations with secluded forests on a Pacific Ocean beach. Registration includes use of conference facilities, deluxe rooms, meals, and nightly wine and cheese parties.

Skip Carter, Conference Chairman   Robert Reiling, Conference Director

---

Advance Registration Required • Call FIG Today 510-893-6784

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

Conference attendee in double room—$395 • Non-conference guest in same room—$280 • Children under 18 years old in same room—$180 • Infants under 2 years old in same room—free • Conference attendee in single room—$525

Forth Interest Group members and their guests are eligible for a ten percent discount on registration fees.

Registration and membership information available by calling, fax or writing to:
Forth Interest Group, PO Box 2154, Oakland, CA 94621, (510) 893-6784, fax (510) 535-1295

Conference sponsored by the Forth Modification Laboratory, an activity of the Forth Interest Group.