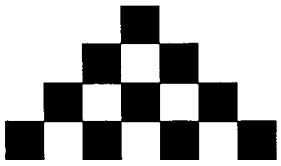
$10 Volume XVII, Number 6 March 1996 April

FORTH DIMENSIONS

Getting to Hardware from Linux

Differential File Comparison

Coordinating Pygmy and C

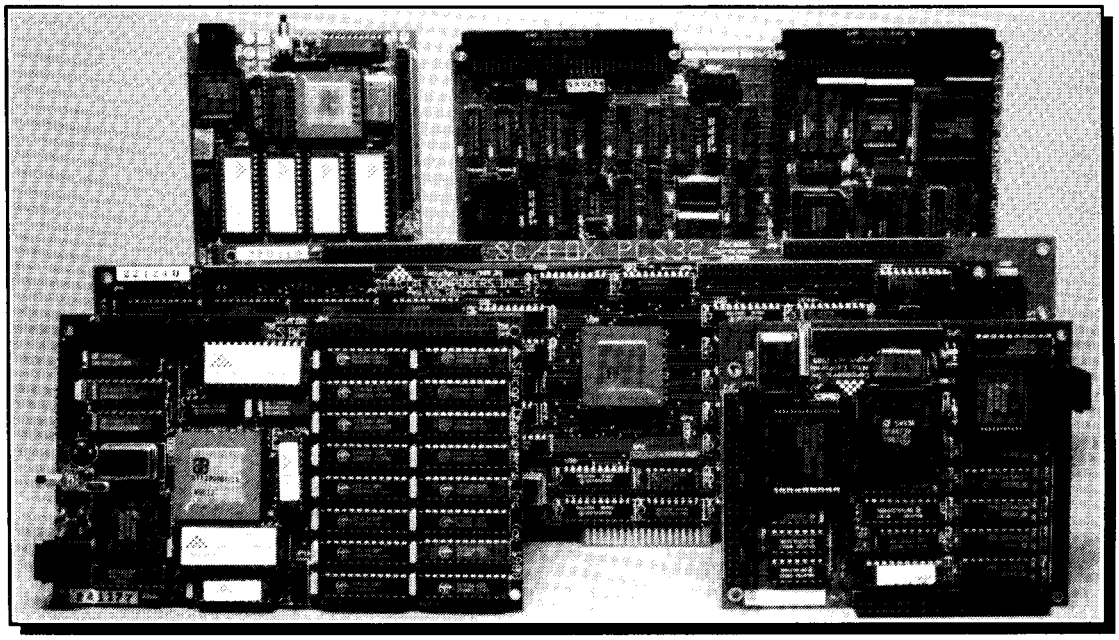


SILICON COMPOSERS INC

FAST Forth Native-Language Embedded Computers

DUP

>R



C@

R>

Harris RTX 2000[tm] 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.

SC32[tm] 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

Features

Departments

Editorial

Forth Dimensions
Volume XVII, Number 6
March 1996 April

Published by the
Forth Interest Group

Editor
Marlin Ouverson

Circulation/Order Desk
Frank Hall

I want to thank Mike Elola—he has long served as our "Fast Forthward" columnist. In that role, he has poked and probed at Forth as we know it. Where he has found weaknesses or omissions, he has not hesitated to point them out. When he has discovered avenues whereby Forth might navigate into wider arenas, he has offered suggestions and road maps.

We have always enjoyed Mike's contributions, especially when they have made us collectively uncomfortable. He often has helped to keep our heads out of the sand by talking about contemporary programming expectations and implementations and interfaces, support of which is expected by people coming to Forth from other contexts.

But this issue contains Mike's last contribution to the column. In it, he shows in anecdotal form where his career path is leading him. And he invites someone new to take over "Fast Forthward." If you would like to write about Forth and contemporary programming issues, about doing Forth business, and/or about Forth success stories, I'd enjoy hearing from you.

Meanwhile, columnists Wil Baden and Skip Carter will continue their challenging and informative columns, "Stretching Forth" and "Forthware."

Working Forth

Another Fortune 500 company has joined the ranks of those pursuing FIG members to fill current openings for Forth programmers. Remember to keep in touch with the FIG office when you are looking for Forth work—it's a pleasure to help our active members find rewarding, interesting jobs.

We Only Ask Two Things...

We look forward to sharing with you the useful, educational, and interesting information that crosses our editorial desktop in the coming year. Two things will make that possible:

First, please keep your membership in the Forth Interest Group current. Members' dues are FIG's financial mainstay, paying the bills that keep the magazine in print and the office open. Our not-for-profit organization needs the support of each of us if it is to stay in good health. You could also encourage your employer to call FIG and inquire about the benefits of a corporate membership.

Secondly, write about your Forth experiences and discoveries, and encourage others to do so. *Forth Dimensions* represents the relevant ideas and work of its readers. A lot of interesting projects deserve a showcase, but we don't know about them until someone brings them to our attention. And fundamental Forth concepts can always be explained again, and perhaps better, in a well-written tutorial.

Your participation will be most welcome—stay in touch!

—Marlin Ouverson, Editor
editor@forth.org

On the Stack...

Coming soon in Forth Dimensions:

Switching DC and AC Electrical Power
R/W DOS Disks from non-DOS Hardware
Safety-Critical Systems
A Simple State-Machine Lexicon
Taming Variables And Pointers

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

Subscription to *Forth Dimensions* is included with membership in the Forth Interest Group at $45 per year ($53 Canada/Mexico, $60 overseas air). For membership, change of address, and to submit items for publication, the address is: Forth Interest Group, P.O. Box 2154, Oakland, California 94621. Administrative offices:
510-89-FORTH Fax: 510-535-1295

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 $45/53/60 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

Send your feedback, questions, criticisms, and other responses to editor@forth.org or to the editor in care of the Forth Interest Group. Submissions may be edited for clarity and length.

List Length and Hash Quality

Xan Gregg (*FD* XVII/4) uses the average list length to evaluate the quality of the hash functions. This neglects the effect that the distribution of the list lengths can have on the performance. Viewed another way, it just tells us how many buckets are filled.

As a better evaluation method, consider the cost of accessing every item present in the table once. For one bucket with a list of length l it is l(l+1)/2 comparisons. So for the whole table it is

$$\sum_{list \in buckets} \frac{length(list)(length(list)+1)}{2}$$

M. Anton Ertl
anton@complang.tuwien.ac.at

Xan Gregg replies:

It's true that the average list length does hide distribution variations, but that is why the provided ANALYZE-HASH function also prints a simple frequency table of list lengths which would help find any distribution anomalies. I should have stated in the article that one should strive to eliminate any particularly long lists before using the average list length metric.

That said, I happily recognize Anton's function as a superior indicator of hash function quality, since it does take distribution variations into account, so any irregularities in distribution will show up in the value. I would, however, convert the summation to an average by dividing by the number of entries hashed, thus providing an "average access" indicator.

Objects of Design

Dear Mr. Ouverson,

I applaud the November-December 1995 issue of *Forth Dimensions*, especially the data-array management issues discussed in the articles on hashing and associative lists. These data array articles stand as an excellent continuation of the recent article by Mr. McGowan covering sets, stacks, and queues. These types of articles have served as both general programming references and as Forth-specific references useful for data-management programming tasks.

The data-array creation articles take an approach that creates arrays within the Forth dictionary. Have any Forth programmers built these structures outside the Forth dictionary—and what tradeoffs occur as a result?

In reference to the ongoing evangelism for object orientation with Forth, the enthusiastic support for object-oriented programming (OOP) seems to overlook the companion design methodology that supports object-oriented programming benefits: class hierarchy design, dynamic object behavior, and functional data flow design. In view of the software quality and scheduling crisis, I hope to encounter OOP articles that review object-oriented design (OOD) methodology applied to Forth programming.

Thank you,
Wil Blake
blakew@sunrise.cse.fau.edu

Opportunists, Blind Disciples, and Fanatics

I read a piece in the November 1995 issue of *Forth Dimensions* entitled "OOP, Forth and the Future."

I disagree. I am not criticizing Mr. Kneusel, as much of what he says is absolutely true, e.g., write a commercial operating system in Forth, support Windows better, develop a good way to link to C libraries (because they are there). Just don't blindly go after OOPs. Let others make the mistakes.

In my opinion, Forth does *not need to* "climb on" any bandwagon or "ride" any tide in the computer world. Let the other guys find out for themselves. Most won't and so what? The computing community is a world full of users, rich opportunists, blind disciples, and fanatics with a very few careful, thoughtful persons sprinkled here and there.

Users buy computer stuff and make the opportunists rich. Nothing wrong there. It's the American way.

The chief fanatics generally are self-proclaimed oracles who collect a sometimes huge following of blind disciples (teachers) and other, ordinary fervent fanatics (followers).

The users and the opportunists *are employing* somewhat logical reasoning. The opportunists say, "I will sell this over-priced stuff because I can." The users say, "I will buy this because I want it; it's 'cheap,' and its available." Both groups are basically happy with their respective lot.

Who are the few careful, thoughtful persons? I hope we, the Forthers, are among them.

Forth is anathema in the computer community because of the extremely, fervently religious nature of so-called "computer science," not because Forth has any flaws or disadvantages at all. In fact, in an atmosphere like this, anything that has a small following and great advantages is all the more to be cast out.

What is the world's most productive language? If you can't answer that, someone must be forcing you to read this. I have used Forth since it was first made available by the little group of devotees in the 70's. I can testify from firsthand experience that Forth is more than 10X more productive (yes, that is at least a ten back there) than any other language you can name.

C is an abominably complex language that lives in an environment that leads to woefully low productivity. C programmers gleefully deride Forthers because Forth uses stacks. And just where do their so "wonderful," "local" variables go, anyway? Use C and find out for yourself.

C++ is even worse—much, much worse. Don't take my word for it. Again, use it and see.

UNIX is an awful, bug-ridden anachronism written in C, proclaimed in leather-bound, gilt-edged tomes (with bound-

in, red ribbon markers) and jealously guarded by a committee of high priests (although you must find a rabbi to tell you how it must be used). Just try to criticize it (of course, after using it) and see how you get sneered at.

What is it that OOPs provide us (other than the obvious literal meaning of the word)? Reusability? No! Information hiding? Those are just words. You can't hide anything, and why should you be proud of trying? Code that is easier to understand? No! Faster execution? Well, maybe a little faster than C. Portability? Absolutely not! A common code-speak to unite the "babble?" No, no, no! Is it cute and complex? Absolutely! Aha, there it is. That must be why it is so runaway popular. I no longer argue with *them*, the believers. As long as someone will pay me to, I will continue to use C, C++, UNIX, X-Windows, and Fortran.

Derision aside for a moment, there is, in my opinion, one useful thing that OOPing has introduced to the world, and that is the *concept* of an object that is an instance of a class. An object can be created on the fly, at run time, and the data owned by any object is always accessible through the object's class methods. That's about it. It's a concept, a way of looking at a problem. Forth can already do this. Use a concept if it makes sense.

Perhaps there are other OOPing advantages I have overlooked. The Forth community should continue to discuss the useful ideas as they spring up.

When the cost to create and maintain code is important, Forth always wins. Use Forth when it affects *your* bottom line. Stick with what's important.

Philip R. Monson
Kekaha, Hawaii

dot-quote

"I was particularly struck by reading the Turing Award lecture of John Backus, in which he complains that programming languages are growing more enormous but not more powerful. He attributes this to the fact that conventional languages are built on a large rigid framework, so trying to make them more powerful entails tacking on more and more features. This makes languages that are bigger and bigger, and harder and harder to use. He proposes a different approach, where a language has a small and flexible framework on which features can be added as changeable parts, rather than built-in. As I read this, it became apparent to me that (classic) Forth offers very much this approach. It would be interesting if someone with a strong background in Computer Science could compare the solution given by Backus with the solution given by Moore. I think that here, too, some important ideas within C.S. were anticipated by Forth.

"As anecdotes about Charles Moore disclose, language is apparently far more malleable, in his eyes, than in the eyes of most people. The impressive thing is that the ability to exercise a high degree of control over the shape of the language itself is not reserved for the inventor alone—it is available to the moderately adept user. There should be C.S. papers which examine the idea of making language itself more malleable and flexible—the approach Moore has taken....

"Charles Moore took a look at the problems of computer programming from the point of view of the programmer rather than that of the theoretical computer scientist. The fact that many of his ideas, developed independently, coincide with or anticipated the findings of mainstream Computer Science is a very significant fact which needs to be played up in the literature. The claims that the Forth community makes about the powers of the language need to be made precise and, at least experimentally, documented. And all this needs to be done by people who know the literature and language of Computer Science."

—John J. Wavrik
jjwavrik@ucsd.edu
Excerpted from comp.lang.forth with permission

Coordinating Pygmy and C

Frank Sergeant
San Marcos, Texas

A few changes allow Pygmy Forth to be wrapped inside a C program. The C program can pass strings to be INTERPRETed by Pygmy. Pygmy can call C library functions or functions defined in the C program.

Overview

The essence of this method is this: The C wrapper program allocates RAM into which it loads the Forth image. The C wrapper contains a table holding the addresses of the functions or structures (variables) we want Forth to be able to access. C calls the loaded Forth image as a function, passing along the address of the table. Via the table, Forth can call the C functions, or read or modify the C structures (variables). Eventually, Forth terminates, returning control to the C wrapper.

The mere mention of the function name in the table in the C program causes the referenced function to be included by the linker when the C program is compiled and linked.

The address of the table is passed to Pygmy on the "command line" that Pygmy will INTERPRET. Indeed, this mechanism is general enough that any Forth code to be INTERPRETed can be passed to Pygmy on this "command line." The same PYGMY.COM file that can serve as the Forth image to be loaded and called by the C program can also be executed directly from DOS, either with or without a string to be INTERPRETed.

The CPU and Operating System

Pygmy runs under DOS on IBM PC-compatible computers with CPUs in the 80x86 family (8088, 8086, 80286, 80386, 80486, Pentium, etc.) Although very widespread, this family of CPUs is so weird that perhaps a word or two of terminology are in order.

Memory is segmented. A full address is composed of two parts. The offset portion indicates how far into the segment the address is, but doesn't indicate where the segment begins. Often, once the segment registers are set up, we forget about them and treat the offset portion as if it were the address. This works fine as long as all the segment registers point to the same segment and we are content to access memory only within this one segment. If we want to access memory outside of our segment, we must consider the segment portion of an address as well as the offset portion. A full address might be written as SEG:OFF. For example 0123:4567 would mean an offset of 4567 hexadecimal into the segment beginning at 0123 hexadecimal times 16. I said it was weird. You can think of the segment portion of an address as supplying the starting position of a segment in terms of paragraphs, where one paragraph is equal to 16 bytes.

In terms of Pygmy, the words @ and ! take 16-bit addresses (the offset only portion) and are relative to the single segment into which Pygmy is loaded by DOS. DOS takes care of setting up the segment registers, and we can more or less forget they even exist—unless we want to access something outside of our segment. In C-for-DOS terminology, a far pointer is a SEG:OFF address. Pygmy can fetch and store from/to such a 32-bit address with the words L@ and L!. We will need this capability to access C data and functions.

The word "address" sometimes means the full SEG:OFF (far pointer) 32-bit address and sometimes just means the 16-bit offset.

Although later members of the 80x86 family of CPUs have various modes (16-bit, 32-bit, protected, virtual, real), for our purposes, they are all just faster versions emulating the original 16-bit 8088/8086 "real mode."

The segment registers are CS, DS, ES, and SS. SS is the stack segment register and is used in connection with the hardware stack pointer SP (which is Pygmy's data stack pointer) and register BP (which is Pygmy's return stack pointer). CS is the code segment register used in connection with the CPU's IP instruction pointer register (program counter). Most data access is in connection with the DS (data segment) register. Pygmy keeps the top item of the data stack in the BX register.

Command-Line Access

Pygmy Forth is a 16-bit, real-mode program loaded from a .COM file. DOS loads it into a 16-bit segment at an offset of $100 (256 decimal). The first $100 bytes are reserved by DOS as the Program Segment Prefix (PSP). Within the PSP (at offset $80), DOS places any command-

line parameters. This takes the form of a Forth-style counted string followed by a carriage return. For example, invoking Pygmy from the command line as

```
C:\>pygmy DUP DUP DUP
```

causes DOS to put the string "DUP DUP DUP" at offset $80.

If Pygmy could interpret from a string, say with the word EVALUATE (a # -), it could interpret the command line with

```
$80 ( a) COUNT ( a #) EVALUATE.
```

We add the word EVALUATE to Pygmy as in Figure One.

Figure One. Adding EVALUATE to Pygmy.

```
$80 CONSTANT COMMAND-LINE
( it acts like a counted string)

: EVALUATE ( a # -)
  >IN 2@ PUSH PUSH    TIB @ PUSH  #TIB @ PUSH
  ( a #) #TIB !  TIB !  (   )
  0 0 INTERPRET
  POP #TIB !  POP TIB !   POP POP >IN 2!  ;
```

We still have one minor problem. If an error occurs, we must restore a proper value to TIB. Otherwise, QUERY will try to use the most recently EVALUATEd string as the terminal input buffer. It might not be long enough, etc. So, we also change the definition of ABORT slightly. Pygmy already saves the default word to execute for EMIT in the variable DEFAULT-EMIT. This is used inside ABORT so if an error occurs while output is directed to the printer, ABORT directs the output back to the screen. Similarly, we create a DEFAULT-TIB so if an error occurs while EVALUATEing a string, the value of TIB can be restored to its proper value. [See Figure Two.]

Figure Two. Restoring TIB after an EVALUATE error.

```
VARIABLE DEFAULT-TIB
TIB @ DEFAULT-TIB !

: (ABORT  ( -)
  ['] DEFAULT-EMIT 1+ @  ['] EMIT 1+ !
  DEFAULT-TIB @ TIB !
  HERE TYPE$ SPACE POP POP TYPE$ SP! BLK @ ?DUP DROP
QUIT ;

' (ABORT IS ABORT
```

Now Pygmy can INTERPRET any string passed on the command line, e.g.,

```
C:\>pygmy  ." Hello, how are you?" CR
KEY DROP
```

or

```
C:\>pygmy  WORDS BYE
```

If there is no string passed on the command line, an empty string is built at offset $80. EVALUATEing it is equivalent to a NOP. We might as well EVALUATE the command line every time Pygmy is started. This is easily done by adding COMMAND-LINE COUNT EVALUATE to BOOT, as in Figure Three.

Easy C

Once we can evaluate the command line, we can write a C program that executes Pygmy with C's spawn call which passes a command line to Pygmy. When Pygmy does BYE, control is returned to the C program.

This is the simplest way to combine Pygmy and C, as it does not require modifying Pygmy any further than described above. However, it has the disadvantage of not allowing Pygmy's dictionary to be extended easily. Each time C calls (spawns) Pygmy, the original Pygmy executable file is reloaded. In other words, the Forth image does not stay resident in memory. Pygmy won't remember anything between calls. (However, if all we want to do is allow Pygmy access to C functions or C library routines, this may be sufficient.)

A little further work on the kernel cures those problems.

A More Complete C/Forth Interface

The next step modifies the Pygmy kernel so it can be called as a subroutine. The C wrapper program builds a table of addresses to be passed to Pygmy. C allocates an array of bytes named loadbuffer, opens the file containing the Forth image (e.g., PYGMYC.COM), then reads the file into the loadbuffer beginning at an offset of $100 (as expected by a DOS .COM file). Once this housekeeping is finished, C can call Forth as often as it wishes as long as it builds a "command line" at an offset of $80 into the loadbuffer array. Typically, the first thing we do is pass the address of the table to Forth.

There are still a few details to tend to, both in the Forth kernel and in the C wrapper.

Making the Forth Kernel a Subroutine

We play fast and loose with the stacks when fooling around interactively in Forth. A minor underflow or overflow of Forth's data stack isn't usually a problem, but we don't want to screw up C's stack. Therefore, upon entry to Forth, we carefully preserve the registers and C's stack pointer. Then we set up private stacks for Forth in its single 16-bit segment. We define #BYE to restore C's registers and stack pointer, and do a long return to the C wrapper while passing back a return code.

We've got another question to decide. Do we want to be required to execute the Forth from a C wrapper, or do

Figure Three. Interpreting the command line when BOOTing.

```
 ( Change BOOT to interpret the command line)
: (BOOT ( -)
  COMMAND-LINE COUNT ( a #) EVALUATE
  ( if it doesn't do BYE, we fall through to the following:)
  $3F ATTR !  CLS
  CR ." PYGMY Forth v1.4  modified to interpret command line"
  OPEN-FILES      .FILES
  CR  ." hi"     QUIT ;

' (BOOT IS BOOT
```

we want the option of executing it as a .COM file directly from the command line? By leaving the original definition of BYE which returns to DOS and defining an alternate #BYE which returns from subroutine, we can use the same PYGMY.COM either as a Forth image inside a C program or as a stand-alone executable at the DOS command line.

The main changes to the kernel are

1. Set up three slots near the beginning of the image (for saving the calling program's SS and SP and for holding the initial Forth word to be executed).
2. Save the registers upon entry. This is necessary if we wish to return to the C program, but doesn't hurt anything if we return directly to DOS.
3. Force the other segment registers to have the same value as CS. Again, this is necessary if Pygmy is called from C, but doesn't hurt anything if Pygmy is executed from DOS.
4. Define #BYE to return to C rather than to DOS.

Take a careful look at the definition of boot. [Fig. Four] The first instruction is an unconditional jump to skip over six bytes, which are filled with zeroes. These six bytes form the three slots to be used to save the calling program's stack pointer segment and offset (i.e., SS and SP) and for holding the address of the initial word for Pygmy to execute (typically RESET).

As PYGMY.COM is a .COM file, the image will be loaded at an offset of $100. Since the jump instruction takes two bytes, we now know the addresses of the three slots: SP is saved in the first slot at address $102. SS is saved in the second slot at address $104. boot expects to find the address of the first word to be executed in the third slot, at address $106.

After the jump instruction and the three slots, the PUSHALL, macro pushes a lot of registers on C's stack, the value of CS is put into the other segment registers, and C's stack pointer registers are saved in the first two slots. Interrupts are disabled prior to setting up the Forth stacks, then restored to their previous state (enabled or disabled). Finally, a jump is made to the address stored in the third slot, and we are away—running Forth.

Figure Four. Defining boot.

```
CODE boot
   HERE 8 + JMP,          ( jump over the next 6 bytes          )
   0 ,                    ( first slot, at $102, for saving SP  )
   0 ,                    ( second slot, at $104, for saving SS)
   0 ,                    ( third slot, at $106, for holding    )
                          (   address of RESET                  )
   PUSHALL,               ( save registers on C's stack         )
   CS PUSH, DS POP,       ( copy CS to DS so that the three     )
                          (   slots will be addressable         )
   SS AX MOV,
   AX $104 ) MOV,         ( save C's stack segment register     )
   SP $102 ) MOV,         ( save C's stack offset register      )
   PUSHF,  BX POP,        ( save interrupt status in BX         )
   DS AX MOV,
   AX ES MOV,             ( copy CS & DS to ES                  )
   CLI,                   ( disable interrupts                  )
   AX SS MOV,             ( copy CS to SS                       )
   RSTACK #, BP MOV,      ( initialize return stack             )
   DSTACK #, SP MOV,      ( initialize parameter stack          )
   BX PUSH,
   POPF,                  ( restore interrupt status            )
   $106 ) AX MOV,
   AX JMP,                ( jump to RESET                       )
END-CODE
```

Figure Five. Restoring registers for the return to C.

```
CODE #BYE ( returncode -)  ( use this for returning to C)
  PUSHF,          ( put flags on stack for popping into AX    )
  AX POP,         ( save interrupt status in AX               )
  CLI,            ( disable interrupts while we change stacks )
  $102 ) SP MOV,  ( restore C's stack pointer register)
  $104 ) SS MOV,  ( restore C's stack segment register)
  AX PUSH,        ( put saved flags back on the stack )
  POPF,           ( restore interrupt status)
  AX POP,         ( remove value AX saved by PUSHALL, in boot )
  BX PUSH,        (  then replace it with return code)
  POPALL,         ( restore C's registers except AX  )
  LRET,           ( far return to C with return code in AX    )
END-CODE
```

Figure Six. New macros clean up other definitions.

```
( extend assembler with some macros)
: PUSHALL, ( -) DS PUSH, ES PUSH, SI PUSH, DI PUSH, BP PUSH,
  DX PUSH, CX PUSH, BX PUSH, AX PUSH,  ;

: POPALL, ( -) AX POP, BX POP, CX POP, DX POP, BP POP,
  DI POP, SI POP,  ES POP, DS POP,  ;
```

Similarly, we define a matching #BYE to restore the saved registers and do a far return back to C. [Figure Five]

We can then return to C with a return code of zero by typing 0 #BYE.

The macros PUSHALL, and POPALL, which simply push or pop a bunch of registers, remove some of the clutter from the new definitions of boot and #BYE. [Figure Six]

The last block of the kernel is responsible for plugging the address of RESET into boot. Previously, this was plugged directly into a move immediate instruction at seven bytes into the definition of boot. We must change this seven to a six to plug the address of RESET into the third slot in boot.

That does it, as far as the changes to Pygmy's kernel go. Regenerate a Pygmy kernel with 1 LOAD and you are done. There are still a few additions needed to let Pygmy call the C functions, but they do not need to be part of the kernel. In particular, we need to be able to do a long call (an intersegment indirect call), so we add a definition for LCALL, to the assembler. Then we define some macros and defining words to make it easy to access the C functions and variables from within Pygmy. These will be shown later in an example of calling Borland graphics routines from Forth.

Alignment

Under DOS, a .COM file is given its own 16-bit segment. All four segment registers (CS, DS, ES, SS) hold the same value. The first $100 bytes of the segment are reserved for the PSP (Program Segment Prefix). The actual executable code begins at an offset of $100 into the segment. The first byte of the file is loaded at address (offset) $100, the second byte of the file is loaded at address $101, etc.

The way segment:offset addressing works in real mode, the physical address of the beginning of the segment must be evenly divisible by 16. This is because the physical address of the start of a segment equals 16 times the value in the segment register.

When the C program allocates memory to hold the Forth image, it returns a pointer made up of a segment value and an offset. We cannot be sure exactly which physical address will be returned. (Actually, under the "huge" memory model, we know the offset will be less than 16.) If the offset is not zero (or at least divisible by 16), the block of memory allocated by C is not on a segment boundary. We must walk the pointer forward until we reach a physical address that is divisible by 16. Once we have the physical address for the beginning of the segment, we convert the physical address to an equivalent segment:offset form where the offset portion is zero. This is the beginning of the segment and is also the PSP. We add $100 to this address to find the address into which we copy the PYGMY.COM image. This is also the address C will use when calling Forth as a subroutine.

When DOS loads a .COM file, it forces all four segment registers to hold the address of the segment. But the call by C to SEG:0100 only forces the correct value into the CS register. It doesn't alter the other three segment registers. Thus, we must copy the value in CS into DS, ES, and SS. The earlier description of boot showed how we set up the segment registers. The important point here is that the call C makes to Forth must use the correctly aligned address and the segment:offset form of this address. The code of LP.C in Listing One shows how we do this. Note, this trickery is fine in real mode, but would require a different

Figure Seven. Declaring void pointers.

```
void *pointers[] =
{
     (void *) &j,                    // integer      0
     (void *) tst,                   // function     1
     (void *) initgraph,             // function     2
     (void *) closegraph             // function     3
     };
```

approach in protected mode.

Memory Model

The C used in this project is Borland C/C++ version 4.5, creating a "huge" model DOS executable. We pick huge instead of large so the C routines will set the DS register correctly upon entry, rather than assuming DS is already correct. Since we want to devote an entire 16-bit (64 Kbyte) segment to Pygmy, the large or huge model makes the most sense. To match the huge model, we use a far return instruction (`LRET,`) when `#BYE` returns to the C program.

How C Calls C Routines

C can call subroutines in various ways. The default method in Borland C's huge model is to push the subroutine's arguments onto the stack from right to left, do a long call (intersegment call) to the subroutine, then clean up the stack after the subroutine returns. For example, for a C function prototyped as

```
int add (int a, int b);
```

upon entry to the subroutine 'add' the stack looks like this (two bytes for each item):

```
value of b
value of a
segment of return address
offset of return address <-- top of stack
```

Thus, when Forth calls the same C subroutine, it must set up the stack the way C expects to find it. If we define the Forth word `add` to call the C subroutine `add`, we want to write the arguments in the same order. That is, if the C prototype shows `int a` on the left and `int b` on the right, then from Forth we want to say

```
a b add
```

and *not* have to rearrange this to

```
b a add
```

In other words, we want our stack-effects comment in Forth to look like the C prototype, at least as far as the order of the parameters is concerned. However, this puts the values on Forth's data stack in this order:

```
value of a
value of b
segment of return address
offset of return address <-- top of stack
```

We could fix this with a `SWAP`, if we only had two arguments, but such a solution becomes more awkward with more than two arguments. The solution I have adopted is to push the arguments from the data stack to the return stack, then switch BP and SP so Forth's return stack becomes C's stack for the purpose of calling the C routine. Thus, we easily reverse the arguments into the form expected by the C subroutine while preserving the readability of our source code.

We could define a separate `CODE` word in Forth, by hand, for every C routine to be called, but this would get rather tedious and error prone. Instead, we create a special defining word to build these routines for us. Then we can define the Forth words that call the C routines with a simple list of index values and counts of input and output arguments.

16-bit DOS C routines return zero, one, or two words. A one-word result is returned in AX. A two-word result (such as a long int or a far pointer) is returned in DX:AX. Forth needs to know how many words of result to expect, so these values can be pushed onto the Forth data stack.

Note that this describes the default calling conventions used by Borland C. Other calling conventions are possible. It is important that you make the Forth and C agree on the calling convention to be used. An easy way to do this in DOS is to write a short program with a small dummy subroutine, such as

```
main (void) {
     int j;
     int test (int a, int b, int c);

     j = test  (1, 2, 3);
}

int test (int a, int b, int c) {
     return a+b+c;
}
```

Then, compile and link this dummy program in the memory model you want to use (such as "huge"), with debugging turned on. Then, examine the assembly language produced. Bring up the program in the debugger and set a breakpoint at `test` and step through it instruction by instruction, examining the stack. You will quickly see exactly how the parameters are passed.

The Table

The table is simply a listing of the addresses of the subroutines and variables in the C program (or in C libraries). A C function name without the ending parenthesis marks returns the address of the function. Thus, in the following example, we do not need to put an ampersand before the function names. On the other hand, a variable name returns the variable's value, rather than its address. We must precede the names of variables with the ampersand in order to put the address of the variable into the table. In the following example, `j` is a variable (an integer) and `tst()`, `initgraph()`, and `closegraph()` are functions. Since we declare pointers to be an array of void pointers, we cast each address to be a void pointer. [Figure Seven]

Figure Eight. Installing the pointer table's address.

```
// install the address of the pointer table
sprintf(forthstr, " %d %d  CTABLE 2! \r",
                FP_SEG(pointers), FP_OFF(pointers) );
strcpy(commandline+1, forthstr);
commandline[0] = (char) (strlen(forthstr)-1);
exit_status = forth();
```

Because we are using the huge memory model, each pointer in the table consists of a segment and an offset, and has a length of four bytes. We must pass the address of pointers to Forth. Then Forth can look up the address for each of the items in the table. The C program passes the address of pointers the first time it calls Forth. Since Forth will attempt to INTERPRET the string at offset $80 in its segment, we place a string at that address which will set the four-byte variable CTABLE to the correct value. First, we build the string in forthstr using the sprintf function. Then, we move this string to address $81 in the Forth segment. Then we plug the length of the string into address $80. [Figure Eight]

Note that FP_SEG extracts the segment value from a far pointer and FP_OFF extracts the offset value from a far pointer. Suppose, in the above, that the address of pointers is 0123:4567. The above example would compose this string

```
291 17767  CTABLE 2!
```

and place it where PYGMY.COM expects to find its command-line parameters. (Note that $0123 is 291 decimal and $4567 is 17767 decimal.) After that, Forth can access the addresses in the pointers array via the value of its CTABLE variable.

Calling C Routines from Forth

To get the address of a C function or variable, we need to know its position in the table. In the above example, the address of initgraph() has an index of 2. Each entry is four bytes long, so we get the address of the table and add eight bytes (two times four) to it and do an L@.

In this approach, we do not need to know the address of the function, nor even the address of the pointers table, when we define the Forth words that will call the C routines. All we need to know are the indexes of the routines in the table. We look up the actual address at run time.

It would make for a faster call if we knew the actual addresses at the time we define the Forth words. This could be done by having the C program call Forth (probably once for each routine) with the address of each routine, extending Forth's dictionary. The startup would be a little slower, but each call would be faster. If this were done, the Forth code would not need to know the indexes of the items in the table. Thus, the position of the functions in the table could be changed without needing to change the Forth. While this can be faster, it is a little more complicated, so we will stick with the simpler method for now.

For each C routine to be called, we define a separate Forth CODE word with CDECL:. In reading the definition of CDECL: note that the words that lay down assembly code end in a comma. Thus, BX PUSH, lays down the code to push the BX register to the stack pointed to by SP, but PUSH without a comma is what most Forths call >R. SWITCH, is a macro that lays down the code to switch the contents of Forth's data stack pointer and return stack pointer (it exchanges SP and RP). This is handy because the hardware PUSH, and POP, instructions work only on the stack pointed to by SP. SWITCH, allows us to use PUSH, and POP, on either stack. PUSH-ARGS, is a macro that takes the count of input parameters and lays down the code to move each of the parameters from Forth's data stack to Forth's return stack. GET-RESULT, is a macro that takes the count of result values and lays down the code to move the result, if any, from the register(s) C uses to Forth's data stack.

The only tricky part may be keeping track of what is done at compile time when CDECL: creates a new Forth word, versus what is done later at run time when the execution of that Forth word sets up the parameters in the form C expects to find them, does a far call to the C routine, then cleans up the stack and puts any result on Forth's data stack.

Pygmy's assembler doesn't have an instruction to do a far call (an intersegment indirect call), so code for the LCALL, instruction needs to be added to Pygmy's assembler. Put the following definition on block 131 or so. Some of the support words, such as R>M, are headerless, so the entire assembler must be reloaded.

```
: LCALL,    ( mem | reg  -)
  1REG?  IF R>M  THEN
  $FF C,  $18 OR modDISP,    ASM-RESET ;
  ( eg   0 [BX] LCALL,  or   DX LCALL, )
```

Accessing C Variables From Forth

We have a similar defining word for C variables. CVAR: defines a Forth word that returns the segment:offset address of the C variable. All that needs to be known at compile time is the index value for the variable in the pointers array in the C program.

The Table as Seen From Forth

The Forth code in Figure Nine uses CDECL: and CVAR: to define Forth words to access the C integer j and to call the C routines tst(), initgraph(), and closegraph().

Note that the counts of input and output parameters are needed only for the functions and not for the variables. Also, the counts of input and output parameters are actually the number of 16-bit words involved. For example, tst() takes two 16-bit integers and returns one 16-bit integer. On the other hand, initgraph() takes only three input parameters, but each is a four-byte far pointer, hence the value 6. Three 32-bit values is expressed as six 16-bit values for the purpose of moving the

Figure Nine. Accessing a C integer and calling C routines.

```
( Define the data structures and subroutines)
( index  #in #out)
      0                CVAR:  J              ( - seg off)
      1    2    1     CDECL: TST            ( a b - c)
      2    6    0     CDECL: initgraph      ( Ldrv Lmode Lstr -)
      3    0    0     CDECL: closegraph     ( -)
```

values from Forth's data stack to C's parameter stack.

Putting It All Together Graphically

The listings for BGI.SCR [Listing Two, page 16] and LP.C [Listing One] show an example of how a C wrapper gives Pygmy access to Borland's BGI routines. You can trade off which parts you put in the C wrapper and which parts you define in Forth. For example, rather than keep track of the value of the manifest constants such as DETECT, TRIPLEX_FONT, and HORIZ_DIR in Forth, C variables are created and set to those values. Then Forth can access them via the pointers table.

```
gdriver  = DETECT;
textfont = TRIPLEX_FONT;
textdir  = HORIZ_DIR;
```

Once the C wrapper turns control over to Forth, we can exercise the graphics routines from the keyboard.

Summary

This is meant to be a quick answer to the question "How can I access C routines from Pygmy?" It hasn't gone into the question of "Should this be done?" It hasn't attempted to say what the best method might be. If you need to access Forth from C or C from Forth, here is one way to do it with Pygmy.

I like the idea of being able to call Pygmy as a subroutine from another program as well as executing it from the command line. I believe I will make this part of the version 1.5 I am working on.

Frank Sergeant's permanent e-mail address is pygmy@pobox.com. He also has an FTP site: ftp.eskimo.com/~pygmy "...where I plan to put Pygmy 1.5 when (if) it is ready. I have just placed cpyg.zip on that site. It contains pygmyc.com, lp.c, bgi.scr, and bgi.dow in case you want to experiment with this without modifying the kernel yourself."

Frank began programming in Forth at the age of three after receiving a Forth tricycle for Christmas. He was wealthy once, but spent his fortune on booze, women, and RAM and wasted the rest. He lives in Texas with his beautiful, intelligent girlfriend Beth, who writes his "author's biographies."

Listing One. LP.C

```
/*  lp.c   "Load Pygmy"    by Frank Sergeant     pygmy@pobox.com

      This is an example of co-ordinating Pygmy with C, to allow Pygmy to call C
      subroutines and access C variables.

      To compile and link lp.c with Borland C/C++ version 4.5 from the command
      line, use the following command:
            bcc -v -1- -mh -p- -tDe lp.c graphics.lib

      The options say to turn on debugging, to restrict the instructions to that
      of an 8086, to compile for the "huge" model, and to create a DOS .EXE file
      that also links in the graphics library.

      This program is an example of how to make C library functions available to
      Forth.  Simply put the names of the desired functions into the pointers
      table but without the ending "()".  Similarly, the addresses of C structures
      may be placed in the table.

      This example shows how Borland's BGI (graphics) routines may be used by Forth.
*/

#include <stdio.h>
#include <string.h>
#include <process.h>
#include <dos.h>
#include <graphics.h>
```

(Continues.)

```
#include <stdlib.h>
#include <malloc.h>
#include <fcntl.h>
#include <io.h>
#include <conio.h>

// declare a function to be defined in this file
int tst (int a, int b);

// Declare several variables whose addresses will be passed to Forth.  All except
//    j have some use in connection with the graphics functions.
int j;
int gdriver, gmode, textfont, textdir;

main (int argc, char * argv[]) {

  int handle, file_size, temp;
  char * filename = {"PYGMYC.COM"};      // default file name

  int exit_status;

  char *realbuffer;     // Allot a 64K byte buffer to hold the Forth image.
  char *loadbuffer;     // Adjust this pointer to a proper SEG:OFF where the
                        //  offset is zero.  It will point to the first byte
                        //  in realbuffer that is paragraph aligned.
  char forthstr[80];    // Staging area for Forth command strings
  char *commandline;    // will point to 0x80 into loadbuffer
  int (*forth)(void);   // declare forth to be a pointer to a function
                        //  that returns an integer.

  // This is the Table that Forth's CTABLE variable will point to.
  //  Note the index values in the rightmost column in the comments.
  void *pointers[] =
  {
       (void *) &j,                          // integer        0
       (void *) tst,                         // function       1
       (void *) initgraph,                   // function       2
       (void *) closegraph,                  // function       3
       (void *) getgraphmode,                // function       4
       (void *) setgraphmode,                // function       5
       (void *) restorecrtmode,              // function       6
       (void *) graphdefaults,               // function       7
       (void *) graphresult,                 // function       8
       (void *) getmaxcolor,                 // function       9
       (void *) setcolor,                    // function      10
       (void *) putpixel,                    // function      11
       (void *) line,                        // function      12
       (void *) linerel,                     // function      13
       (void *) lineto,                      // function      14
       (void *) moverel,                     // function      15
       (void *) moveto,                      // function      16
       (void *) circle,                      // function      17
       (void *) settextstyle,                // function      18
       (void *) outtext,                     // function      19
       (void *) outtextxy,                   // function      10
       (void *) &gdriver,                    // integer       21
       (void *) &gmode,                      // integer       22
       (void *) &textfont,                   // integer       23
       (void *) &textdir                     // integer       24
  };
```

```
  // allocate memory for the Forth segment
  realbuffer = malloc(65535);

  // Walk forward in that segment until a paragraph (16 byte) boundary is reached.
  temp = FP_OFF(realbuffer);
  if ( temp % 16 )                        // if not paragraph aligned,
     temp +=  16 -  (temp % 16);          //   then align it
  loadbuffer = MK_FP ( FP_SEG (realbuffer) + temp / 16, 0 );

  forth = (void *) (loadbuffer + 0x100);  // entry point is + 256 bytes
  commandline = loadbuffer + 0x80;        // command line is + 128 bytes

  // collect alternate Forth image file name from the command line, if present
  if (argc == 2) strcpy(filename, argv[1]);

  // load the Forth image into the segment allocated for it
  handle = open( filename, O_RDONLY | O_BINARY);
  if ( handle != -1) {
     file_size = filelength(handle);
        read ( handle, loadbuffer+0x100, file_size);
        close (handle);
  }
  else
      puts("File could not be opened\n");

  // set up some variables Forth will use when calling graphics routines
  gdriver  = DETECT;
  textfont = TRIPLEX_FONT;
  textdir  = HORIZ_DIR;

  // Build a string at forthstr which will be INTERPRETed by Forth
  //   to make Forth's CTABLE variable hold the address of the pointer table.
  sprintf(forthstr, " %d %d  CTABLE 2! \r",
                  FP_SEG(pointers), FP_OFF(pointers) );

  // Copy that string where Forth expects to find a "command line"
  strcpy(commandline+1, forthstr);

  // Plug in the length of the "command line"
  commandline[0] = (char) (strlen(forthstr)-1);

  // Call Forth as a C subroutine
  exit_status = forth();

  printf("The returned exit status is %d\n", exit_status);
  printf("The value of j is %d\n", j);

  free (realbuffer);
  return;
}

// Following is a dummy function to allow us to inspect
//   the exact calling protocol used by C
int tst (int a, int b) {

   static int k = 7;
   int c;

   c = a + b + k++;
   return c;
}
```

(Listing Two begins on next page.)

Figure Two. BGI.SCR

```
scr #  4001
( Load block)
4002 4009 THRU
SAVE PYGMYC.COM


EXIT
Experiments with co-ordinating C and Forth

Copyright 1995 Frank Sergeant
               809 W. San Antonio St.
               San Marcos, TX  78666

               pygmy@pobox.com    (permanent email address)
```

```
scr #  4002
( CTABLE holds the address of the table of addresses in C)

: 2VARIABLE ( -) ( - a)  VARIABLE 0 ,  ;

2VARIABLE CTABLE        ;
```

```
scr #  4003
( Macros to make building a call to a C routine easy)

: PUSH-ARGS, ( #in -)
  ( move parms to Forth's return stack, ie C's stack to be)
  FOR   BX POP,  SWITCH,  BX PUSH, SWITCH,  NEXT ( off seg)  ;

: GET-RESULT, ( #out -)
  DUP 0= IF  DROP  BX POP, ( will refill TOS) EXIT THEN
  2 = IF DX PUSH, THEN AX BX MOV, ( result to TOS)      ;
```

```
scr #  5001
( Load block)
```

```
scr #  5002

2VARIABLE  creates a 4-byte variable that can hold a
           C "far pointer"


CTABLE     holds a seg offset (far pointer) address of the
           table in the C wrapper.  The table contains
           the far pointers to various C functions and
           variables
```

```
scr #  5003
( Macros to make building a call to a C routine easy)

PUSH-ARGS,  is an assembler macro that lays down the code
            to move parameters from Forth's data stack
            to Forth's return stack.  When the C routine
            is called, Forth's return stack will be used as
            C's parameter stack.

GET-RESULT, is an assembler macro that lays down the code
            to move the result (if any) returned by the
            C function from register AX or registers DX:AX
            to Forth's data stack.
```

scr # 4004

```
( Use this to define a word which calls a C routine)


: CDECL:  ( index #in #out -)   CODE    SWAP
  BP AX MOV,  ( save Forth's rstk ptr)
  BX PUSH,    ( put TOS on real stack)    PUSH-ARGS,
  AX PUSH,  SI PUSH, DS PUSH,  ( save important registers)
  SWITCH,  ( make return stack become C's stack)
  CTABLE #, DI MOV,  0 [DI] DI LDS,
  ( now DS:DI points to 1st table entry in the C program)
  SWAP ( #out index) 2* 2* [DI] LCALL,  ( call via the table)
  SWITCH,  ( give us back our real data stack pointer)
  DS POP, SI POP, BP POP, ( & original return stack pointer)
  ( #out) GET-RESULT,  NXT,   ;
```

scr # 4005

```
( Interface to a C data structure)
( Defines a word which returns address of a C data structure)

: CVAR: ( index -) ( - seg off)
  CREATE 4 * ,              ;
   DOES> @ CTABLE 2@ ROT + 2DUP 2 + L@ ROT ROT L@  ;
```

scr # 4006

```
( Define the data structures and subroutines)
( index  #in #out)
     0                CVAR:  J              ( - seg off)
     1    2    1     CDECL: TST            ( a b - c)
     2    6    0     CDECL: initgraph      ( Ldrv Lmode Lstr -)
     3    0    0     CDECL: closegraph     ( -)
     4    0    1     CDECL: getgraphmode   ( - mode)
     5    1    0     CDECL: setgraphmode   ( mode -)
     6    0    0     CDECL: restorecrtmode ( -)
     7    0    0     CDECL: graphdefaults  ( -)
     8    0    1     CDECL: graphresult    ( - result)
```

(Continues on next page.)

scr # 5004

CDECL:

This is a defining word that builds a CODE word to call a C function. The index is used at run time to look up the address of the C function in a table in the C program (the address of the table is stored in CTABLE). The #in parameter parameter tells CDECL: how many 16-bit words to move from the data stack to C's stack prior to calling the C function. The #out parameter tells CDECL: how many 16-bit words to move from AX or DX:AX to Forth's data stack after the C function returns.

scr # 5005

CVAR:

This is similar to CDECL: but is used for variables instead of functions. It defines a word that returns the seg offset address of a C variable. For example, if the 0th item in the table is an integer j,

```
0 CVAR: J
```

defines the Forth word J which will return the seg and offset of the C integer j. J L@ would return the current value of that integer. 17 J L! would change the value to 17.

scr # 5006

This block and the next define words to access all of the variables and functions in the C program's pointers table. These two blocks and the C program's pointers table must be kept synchronized!

J accesses the C integer j just to illustrate the mechanism. The sample program displays the value of j upon termination. You could play with setting it to different values in Forth, then see what value C displays upon termination:

```
17 J L!     0 #BYE
```

(Continues on next page.)

```
      9    0    1    CDECL:  getmaxcolor    ( - u)
     10    1    0    CDECL:  setcolor       ( color -)
     11    4    0    CDECL:  putpixel       ( x y color -)
     12    4    0    CDECL:  line           ( x1 y1 x2 y2 -)
( NOTE: "L" as in Ldrv, Lstr, etc means a 4byte address )
```

scr # 4007

```
( Define the data structures and subroutines)
( index  #in #out)
     13    2    0    CDECL:  linerel        ( dx dy -)
     14    2    0    CDECL:  lineto         ( x y -)
     15    2    0    CDECL:  moverel        ( dx dy -)
     16    2    0    CDECL:  moveto         ( x y -)
     17    3    0    CDECL:  circle         ( x y radius -)
     18    3    0    CDECL:  settextstyle   ( font dir size -)
     19    2    0    CDECL:  outtext        ( Lstr -)
     20    4    0    CDECL:  outtextxy      ( x y Lstr -)
     21                CVAR:  gdriver        ( - seg off)
     22                CVAR:  gmode          ( - seg off)
     23                CVAR:  textfont       ( - seg off)
     24                CVAR:  textdir        ( - seg off)
```

scr # 4008

```
: .ERR  ( -) graphresult .  ;
: Lnull ( - long-null-C-string)   "  "  2 +  CS@  ;
: MSG ( - off seg) " HELLO! " 1+  CS@  ;
: INIT-GPH ( -)   gdriver SWAP  gmode SWAP
  " G:\BC45\BGI\" 1+ CS@  initgraph  ( .ERR ) ;
: TXT ( -)  restorecrtmode ( .ERR ) ;
: GPH ( -)  getgraphmode ( u) setgraphmode  ;
: TXTSIZE ( # -) PUSH textfont L@ textdir L@ POP settextstyle ;
: TST1 ( -)  INIT-GPH
  20 30 moveto  " This is Pygmy! " 1+ CS@ outtext    2 TXTSIZE
  20 40 moveto  " This is Pygmy! " 1+ CS@ outtext    4 TXTSIZE
  20 60 moveto  " This is Pygmy! " 1+ CS@ outtext    8 TXTSIZE
  20 80 moveto  " This is Pygmy! " 1+ CS@ outtext
  300 240 75 circle
  KEY DROP  TXT   closegraph   ;
```

```
                32 J L!     5 #BYE       etc.
```

scr # 5007

Perhaps the most important point to notice is that the Forth stack comment shows the parameters in the very same order as the C function prototypes. If in C you would say

```
circle (200, 300, 75);
```

then in Forth you would say

```
200 300 75 circle
```

For more info about these particular graphic functions, see Borland's BGI documentation.

scr # 5008

Various examples of accessing the graphics functions from Forth. Try typing TST1 to see them in action.

Note! It is unlikely that your Borland graphic drivers are on drive G: as mine are. Be sure to change the path in the definition of INIT-GPH before running these examples!

```
scr # 5009

This block contains more examples of accessing the BGI
graphics routines.
```

```
scr # 4009
( draw various circles)

: END-GPH ( -)
  KEY DROP  TXT   closegraph   ;

: CONCENTRIC ( -)
  INIT-GPH   300 240 ( center)
  15 FOR ( x y) 2DUP  I 5 * ( ie radius)  circle  NEXT 2DROP
  END-GPH  ;

: CIRCLES ( -) INIT-GPH  400 200 ( center)
  10 FOR  ( x y) 2DUP 75 circle
          ( x y) -2 I * +UNDER   3 I * +   ( x y)
  NEXT  2DROP  END-GPH  ;
```

16th Annual ROCHESTER FORTH CONFERENCE

OPEN SYSTEMS

June 19–22, 1996
Ryerson Polytechnic University
Toronto, Ontario, Canada

featuring invited speakers

MITCH BRADLEY	CHARLES MOORE
Firmworks Inc.	Computer Cowboys
creator of "Open Firmware"	inventor of the Forth language

The Institute for Applied Forth Research is pleased to present the 16th Annual Rochester Forth Conference, on Open Systems. Since 1981, the Rochester Forth Conference has been a popular forum for researchers, developers, users, and vendors of the Forth language. This year we extend our welcome to all developers of Open Systems for an exchange of knowledge and opinion!

PAPERS • WORKING GROUPS • POSTER SESSIONS • VENDOR EXHIBITS • TUTORIALS

Papers will be presented on all aspects of Forth technology. Special topics include Open Firmware, Plug and Play Systems, Scripting Languages, SGML and HTML, Java, Distributed Computing. There will also be papers and working groups on Forth programming standards (including ANS Forth), embedded and real-time systems, scientific/engineering applications, and education. Tutorials will include: Introduction to Forth, Advanced Forth, Forth Under Windows, Metacompilation, "Open Boot" Firmware, HTML, Java, and TCP/IP. Many of the major vendors of Forth and related technologies will be on hand to demonstrate their products.

CONFERENCE LOCATION

This year, for the first time, the Rochester Forth Conference moves to Toronto, Ontario—the city Peter Ustinov called "New York run by the Swiss." Toronto is a center of high technology, finance, and culture, and one of the world's most popular tourist destinations. Our conference venue at Ryerson Polytechnic University is in the heart of downtown Toronto, and within walking distance of many attractions. Toronto is ideally located for U.S., Canadian, and international travel.

Register Early and Enjoy Lower Fees!

- ❑ US$400 Attendee (US$475 after April 1st)
- ❑ US$200 Full-time student (CAD$200 for a Canadian student)
- ❑ US$120 Spouse

ROOMS

- ❑ US$150 Single for four nights (Wed., Thu., Fri., Sat.)
- ❑ US$200 Double (2 persons, 1 double bed) for four nights
- ❑ US$100 Student single for four nights

SEND REGISTRATIONS TO:
Rochester Forth Conference
Box 1261 Annandale, VA 22003 USA
lforsley@jwk.com
US check, Visa or MasterCard only

For More Information:
Elliott Chapin
24 Monteith St.
Toronto, ON Canada
M4Y 1K7
echapin@interlog.com
416-921-9560

or see our World Wide Web page at
http://maccs.dcss.mcmaster.ca/~ns/96roch.html

SPONSORS:
Microtronix Systems Ltd.
London, Ontario, Canada

PRESENTED IN COOPERATION WITH:
the Southern Ontario Forth Interest Group
and McMaster University

Stretching Forth

STANDARD FORTH

Differential File Comparison

Wil Baden
Costa Mesa, California

A necessary tool for every programmer is a utility to compare files—particularly source files—to find where and how they are different. A prudent programmer will make a copy of a file before modifying it. In the course of making a series of small modifications to fix a misbehaving application, the programmer can easily lose track of just what has been done. Then the file comparison utility can be used to show the changes.

This utility can be used to show the differences between released versions as well.

To do the job right is not a trivial task. The obvious algorithm will sooner than later fail miserably.

The obvious algorithm is to compare lines until a difference is found, then search forward in both files to find where they are the same again.

The trick is not to look for differences but to look for the *longest common subsequence*—the longest set of lines which are the same in both files and in the same order with what's different interspersed. What's left are the differences.

Differential file comparison is the foundation of version-control systems... I intend to present a Forth personal version-control system based on this code.

How to do this is the subject of HUNT, J. W. AND M. D. MCILROY (1976), "An algorithm for differential file comparison," *Computing Science Technical Report 41*, AT&T Bell Laboratories, Murray Hill, New Jersey. It is based on HUNT, J.W. AND T.G. SZYMANSKI (1977), "A fast algorithm for computing longest common subsequences," *Comm. ACM*, vol. 20 no. 5, pp. 350–353.

In 1976 I implemented this using my own code in Fortran II for an 8K, 16-bit word IBM 1130. It has followed me ever since, becoming re-incarnated on each new platform in whatever the language of the moment was.

I even did this in C for Unix because my output format was more useful than that of the Unix tool `diff`.

Some years ago I did it for MacForth. In the present incarnation it is Standard Forth.

Differential file comparison is the foundation of version control systems—SCCS, RCS, SCVS. In a later article in this series I intend to present a Forth personal version control system based on this month's code.

The algorithm is essentially brute force. Read and save one file, then read records from the other file, trying to find with each record a longer common subsequence than you already have.

Potentially this could require M × N line comparisons, where M and N are the number of lines in each file. In real life that never happens.

The time and memory constraints are still too extravagant. So a really slick trick is used. Instead of comparing whole lines, an integer hash value is computed for each line, and the associated hash values are compared. Making believe that every unique line has a unique hash value, we compute a longest common subsequence.

Not until we print do we check whether equal hash values represent identical lines.

In twenty years of use this has hardly ever happened. In the very few times it has, the effect has been negligible. (You can tell that it has happened when an insertion appears just before a deletion.) It's at least seven years since I've seen it happen.

Of course you can force it to happen by using a poor hashing function. However, the hashing function doesn't have to be sophisticated. The one used here works fine with 32-bit or 16-bit arithmetic.

Where I used to work, the Pascal incarnation was used 30 to 200 times a day for ten years, using 16-bit arithmetic. It was used even after the company went to Unix.

How to Use

After loading the program, given two files, named *e.g.*, PROMISES.BAK and PROMISES.4TH:

```
S" PROMISES.BAK" INPUT TO OLD
S" PROMISES.4TH" INPUT TO NEW
DFC
```

Here's an example comparing the source for DFC with a revision.

```
S" DFC.FO" INPUT TO OLD   S" DFC-HERE.FO" INPUT TO NEW   DFC
```

The output is:

```
   1 DEL> ( DFC - Differential File Comparison.  Wil Baden 1976-1996 )
NEW>    1 ( DFC - Differential File Comparison Using HERE  Wil Baden )
   2    2

  11   11
  12 DEL> : BOUNDS     OVER + SWAP ;        ( a k -- a+k a )
  13 DEL>
  14 DEL> : INPUT      R/O OPEN-FILE ABORT" Can't open " ;
  15 DEL>
  16 DEL> : REWIND                          ( fileid -- )
  17 DEL>              0 0 ROT REPOSITION-FILE
  18 DEL>                  ABORT" Sorry, error rewinding file. "
  19 DEL> ;
  20 DEL>
  21 DEL> : PLACE                           ( s . a -- )
  22 DEL>              2DUP >R >R  CHAR+  SWAP CHARS MOVE  R> R> C!
  23 DEL> ;
  24 DEL>
  25   12 : UNDER      ROT DROP SWAP ;      ( a b c -- c b )

  50   37
  51 DEL> 6000 CONSTANT   lcs-space        ( The larger the better. )
  52 DEL> CREATE          LCS   lcs-space CELLS ALLOT
  53 DEL>
NEW>   38 0 VALUE lcs-space        0 VALUE LCS
  54   39 0 VALUE oldlines         0 VALUE newlines

 394  379     ( Differential file comparison. )
NEW>  380     ALIGN HERE TO LCS
NEW>  381     UNUSED 1 CELLS - 1+ ALIGNED 1 CELLS / TO lcs-space
 395  382     read-newerfile   sort-hash-values   mark-hash-classes

 397  384     build-candidate-table   show-differences
NEW>  385     oldlines newlines - 2 - LCS @ - . ." deletions, "
NEW>  386         newlines 1- LCS @ - . ." insertions, "
NEW>  387             LCS @ . ." unchanged " CR
 398  388     OLD REWIND   NEW REWIND

 401  391 \ Procedamus in pace.    Wil Baden    Costa Mesa, California
NEW>  392
NEW>  393 ARGUMENT INPUT TO OLD   ARGUMENT INPUT TO NEW   DFC
```

This shows that in the old file, DFC.FO,

- Line 1 has been replaced.
- Lines 12 through 24 have been deleted.
- Lines 51 through 53 have been replaced by a single line.
- A few new lines have been inserted after lines 394, 397, and 401.The numbers in the first column are the line numbers in the first file. The numbers in the second column are the line numbers in the second file.

Wil Baden is a professional programmer with an interest in Forth. You can get the text for DFC.FO by e-mail to: wilbaden@netcom.com.
NOT should be equivalent to : NOT 0= ;

FIG
MAIL ORDER FORM

HOW TO USE THIS FORM: Please enter your order on the back page of this form and send with your payment to the Forth Interest Group. All items have one price. Enter price on order form and calculate shipping & handling based on location and total.

"Were Sure You Wanted To Know..."

★***Forth Dimensions*, Article Reference** 151 - $4 0#
An index of Forth articles, by keyword, from *Forth Dimensions* Volumes 1–15 (1978–94).

★**FORML, Article Reference** 152 - $4 0#
An index of Forth articles by keyword, author, and date from the FORML Conference Proceedings (1980–92).

FORTH DIMENSIONS BACK VOLUMES

A volume consists of the six issues from the volume year (May–April)

Volume 1 Forth Dimensions (1979–80) 101 - $15

Introduction to FIG, threaded code, TO variables, fig-Forth.

Volume 6 Forth Dimensions (1984–85) 106 - $15

Interactive editors, anonymous variables, list handling, integer solutions, control structures, debugging techniques, recursion, semaphores, simple I/O words, Quicksort, high-level packet communications, China FORML.

Volume 7 Forth Dimensions (1985–86) 107 - $20

Generic sort, Forth spreadsheet, control structures, pseudo-interrupts, number editing, Atari Forth, pretty printing, code modules, universal stack word, polynomial evaluation, F83 strings.

Volume 8 Forth Dimensions (1986–87) 108 - $20

Interrupt-driven serial input, data-base functions, TI 99/4A, XMODEM, on-line documentation, dual CFAs, random numbers, arrays, file query, Batcher's sort, screenless Forth, classes in Forth, Bresenham line-drawing algorithm, unsigned division, DOS file I/O.

Volume 9 Forth Dimensions (1987–88) 109 - $20

Fractal landscapes, stack error checking, perpetual date routines, headless compiler, execution security, ANS-Forth meeting, computer-aided instruction, local variables, transcendental functions, education, relocatable Forth for 68000.

Volume 10 Forth Dimensions (1988–89) 110 - $20

dBase file access, string handling, local variables, data structures, object-oriented Forth, linear automata, stand-alone applications, 8250 drivers, serial data compression.

Volume 11 Forth Dimensions (1989–90) 111 - $20

Local variables, graphic filling algorithms, 80286 extended memory, expert systems, quaternion rotation calculation, multiprocessor Forth, double-entry bookkeeping, binary table search, phase-angle differential analyzer, sort contest.

Volume 12 Forth Dimensions (1990–91) 112 - $20

Floored division, stack variables, embedded control, Atari Forth, optimizing compiler, dynamic memory allocation, smart RAM, extended-precision math, interrupt handling, neural nets, Soviet Forth, arrays, metacompilation.

FORML CONFERENCE PROCEEDINGS

FORML (Forth Modification Laboratory) is an educational forum for sharing and discussing new or unproven proposals intended to benefit Forth, and is an educational forum for discussion of the technical aspects of applications in Forth. Proceedings are a compilation of the papers and abstracts presented at the annual conference. FORML is part of the Forth Interest Group.

1981 FORML PROCEEDINGS 311 - $45
CODE-less Forth machine, quadruple-precision arithmetic, overlays, executable vocabulary stack, data typing in Forth, vectored data structures, using Forth in a classroom, pyramid files, BASIC, LOGO, automatic cueing language for multimedia, NEXOS—a ROM-based multitasking operating system. *655 pgs*

1982 FORML PROCEEDINGS 312 - $30
Rockwell Forth processor, virtual execution, 32-bit Forth, ONLY for vocabularies, non-IMMEDIATE looping words, number-input wordset, I/O vectoring, recursive data structures, programmable-logic compiler. *295 pgs*

1983 FORML PROCEEDINGS 313 - $30
Non-Von Neuman machines, Forth instruction set, Chinese Forth, F83, compiler & interpreter co-routines, log & exponential function, rational arithmetic, transcendental functions in variable-precision Forth, portable file-system interface, Forth coding conventions, expert systems. *352 pgs*

1984 FORML PROCEEDINGS 314 - $30
Forth expert systems, consequent-reasoning inference engine, Zen floating point, portable graphics wordset, 32-bit Forth, HP71B Forth, NEON—object-oriented programming, decompiler design, arrays and stack variables. *378 pgs*

1986 FORML PROCEEDINGS 316 - $30
Threading techniques, Prolog, VLSI Forth microprocessor, natural-language interface, expert system shell, inference engine, multiple-inheritance system, automatic programming environment. *323 pgs*

1988 FORML PROCEEDINGS 318 - $40
Includes 1988 Australian FORML, Human interfaces, simple robotics kernel, MODUL Forth, parallel processing, programmable controllers, Prolog, simulations, language topics, hardware, Wil's workings & Ting's philosophy, Forth hardware applications, ANS Forth session, future of Forth in AI applications. *310 pgs*

1989 FORML PROCEEDINGS 319 - $40
Includes papers from '89 euroFORML. Pascal to Forth, extensible optimizer for compiling, 3D measurement with object-oriented Forth, CRC polynomials, F-PC, Harris C cross-compiler, modular approach to robotic control, RTX recompiler for on-line maintenance, modules, trainable neural nets. *433 pgs*

1992 FORML PROCEEDINGS 322 - $40
Object oriented Forth bases on classes rather than prototypes, color vision sizing processor, virtual file systems, transparent target development, Signal processing pattern classification, optimization in low level Forth, local variables, embedded Forth, auto display of digital images, graphics package for F-PC, B-tree in Forth *200 pgs*

★ These are your most up-to-date indexes for back issues of *Forth Dimensions* and the FORML proceedings.

Fax your orders: 510-535-1295

1993 FORML PROCEEDINGS 323 - $45

Includes papers from '92 euroForth and '93 euroForth Conferences. Forth in 32-Bit protected mode, HDTV format converter, graphing functions, MIPS eForth, umbilical compilation, portable Forth engine, formal specifications of Forth, writing better Forth, Holon - A new way of Forth, FOSM, a Forth string matcher, Logo in Forth, programming productivity. *509 pgs*

BOOKS ABOUT FORTH

ALL ABOUT FORTH, 3rd ed., June 1990, Glen B. Haydon 201 - $90

Annotated glossary of most Forth words in common usage, including Forth-79, Forth-83, F-PC, MVP-Forth. Implementation examples in high-level Forth and/or 8086/88 assembler. Useful commentary given for each entry. *504 pgs*

eFORTH IMPLEMENTATION GUIDE, C.H. Ting 215 - $25

eForth is the name of a Forth model designed to be portable to a large number of the newer, more powerful processors available now and becoming available in the near future. *54 pgs* (w/disk)

Embedded Controller FORTH, 8051, William H. Payne 216 - $76

Describes the implementation of an 8051 version of Forth. More than half of this book contains source listings (w/disks C050) *511 pgs*

F83 SOURCE, Henry Laxen & Michael Perry 217 - $20

A complete listing of F83, including source and shadow screens. Includes introduction on getting started. *208 pgs*

THE FIRST COURSE, C.H. Ting 223 - $25

This tutorial's goal is to expose you to the very minimum set of Forth instructions you need to use Forth to solve practical problems in the shortest possible time. "... This tutorial was developed to complement *The Forth Course* which skims too fast on the elementary Forth instructions and dives too quickly in the advanced topics in a upper level college microcomputer laboratory ..." A running F-PC Forth system would be very useful. *44 pgs*

THE FIRST COURSE, C.H. Ting 223 - $25

This tutorial's goal is to expose you to the very minimum set of Forth instructions you need to use Forth to solve practical problems in the shortest possible time. "... This tutorial was developed to complement *The Forth Course* which skims too fast on the elementary Forth instructions and dives too quickly in the advanced topics in a upper level college microcomputer laboratory ..." A running F-PC Forth system would be very useful. *44 pgs*

FORTH ENCYCLOPEDIA, Mitch Derick & Linda Baker 220 - $30

A detailed look at each fig-Forth instruction. *327 pgs*

FORTH NOTEBOOK, Dr. C.H. Ting 232 - $25

Good examples and applications. Great learning aid. poly-FORTH is the dialect used. Some conversion advice is included. Code is well documented. 286 pgs

FORTH NOTEBOOK II, Dr. C.H. Ting 232a - $25

Collection of research papers on various topics, such as image processing, parallel processing, and miscellaneous applications. *237 pgs*

F-PC USERS MANUAL (2nd ed., V3.5) 350 - $20

Users manual to the public-domain Forth system optimized for IBM PC/XT/AT computers. A fat, fast system with many tools. *143 pgs*

F-PC TECHNICAL REFERENCE MANUAL 351 - $30

A must if you need to know the inner workings of F-PC. *269 pgs*

INSIDE F-83, Dr. C.H. Ting 235 - $25

Invaluable for those using F-83. *226 pgs*

OBJECT-ORIENTED FORTH, Dick Pountain 242 - $37

Implementation of data structures. First book to make object-oriented programming available to users of even very small home computers. *118 pgs*

SCIENTIFIC FORTH, Julian V. Noble 250 - $50

Scientific Forth extends the Forth kernel in the direction of scientific problem solving. It illustrates advanced Forth programming techniques with nontrivial applications: computer algebra, roots of equations, differential equations, function minimization, functional representation of data (FFT, polynomials), linear equations and matrices, numerical integration/Monte Carlo methods, high-speed real and complex floating-point arithmetic. *300 pgs* (Includes disk with programs and several utilities), **IBM**

SEEING FORTH, Jack Woehr 243 - $25

"... I would like to share a few observations on Forth and computer science. That is the purpose of this monograph. It is offered in the hope that it will broaden slightly the streams of Forth literature ..." *95 pgs*

STACK COMPUTERS, THE NEW WAVE 244 - $82

Philip J. Koopman, Jr. (hardcover only)
Presents an alternative to Complex Instruction Set Computers (CISC) and Reduced Instruction Set Computers (RISC) by showing the strengths and weaknesses of stack machines. **Last 5**

STARTING FORTH (2nd ed.), Leo Brodie 245 - $29

In this edition of *Starting Forth*—the most popular and complete introduction to Forth—syntax has been expanded to include the Forth-83 Standard. *346 pgs*

THINKING FORTH, Leo Brodie 255 - $20

BACK BY POPULAR DEMAND. The bestselling author of *Starting Forth* is back again with the first guide to using Forth to program applications. This book captures the philosophy of the language to show users how to write more readable, better maintainable applications. Both beginning and experienced programmers will gain a better understanding and mastery of such topics: Forth style and conventions, decomposition, factoring, handling data, simplifying control structures. And, to give you an idea of how these concepts can be applied, *Thinking Forth* contains revealing interviews with real-life users and with Forth's creator Charles H. Moore. To program intelligently, you must first think intelligently, and that's where *Thinking Forth* comes in. **Reprint of original**, 272pgs

WRITE YOUR OWN PROGRAMMING LANGUAGE USING C++, Norman Smith 270 - $15

This book is about an application language. More specifically, it is about how to write your own custom application language. The book contains the tools necessary to begin the process and a complete sample language implementation. [Guess what language!] Includes disk with complete source. *108 pgs*

WRITING FCODE PROGRAMS 252 - $48

This manual is written for designers of SBus interface cards and other devices that use the FCode interface language. It assumes familiarity with SBus card design requirements and Forth programming. The material covered discusses SBus development for both OpenBoot 1.0 and 2.0 systems. *414 pgs* **NEW**

DISKS: Contributions from the Forth Community

The "Contributions from the Forth Community" disk library contains author-submitted donations, generally including source, for a variety of computers & disk formats. Each file is determined by the author as public domain, shareware, or use with some restrictions. This library does not contain "For Sale" applications. *To submit your own contributions, send them to the FIG Publications Committee.*

FLOAT4th.BLK V1.4 Robert L. Smith C001 - $8
Software floating-point for fig-, poly-, 79-Std., 83-Std. Forths. IEEE short 32-bit, four standard functions, square root and log.
★★★ **IBM, 190Kb, F83**

Games in Forth C002 - $6
Misc. games, Go, TETRA, Life... Source.
★ **IBM, 760Kb**

A Forth Spreadsheet, Craig Lindley C003 - $6
This model spreadsheet first appeared in *Forth Dimensions* VII/1,2. Those issues contain docs & source.
★ **IBM, 100Kb**

Automatic Structure Charts, Kim Harris C004 - $8
Tools for analysis of large Forth programs, first presented at FORML conference. Full source; docs incl. in 1985 FORML Proceedings.
★★ **IBM, 114Kb**

A Simple Inference Engine, Martin Tracy C005 - $8
Based on inf. engine in Winston & Horn's book on LISP, takes you from pattern variables to complete unification algorithm, with running commentary on Forth philosophy & style. Incl. source.
★★ **IBM, 162 Kb**

The Math Box, Nathaniel Grossman C006 - $10
Routines by foremost math author in Forth. Extended double-precision arithmetic, complete 32-bit fixed-point math, & auto-ranging text. Incl. graphics. Utilities for rapid polynomial evaluation, continued fractions & Monte Carlo factorization. Incl. source & docs.
★★ **IBM, 118 Kb**

AstroForth & AstroOKO Demos, I.R. Agumirsian C007 - $6
AstroForth is the 83-Std. Russian version of Forth. Incl. window interface, full-screen editor, dynamic assembler & a great demo. AstroOKO, an astronavigation system in AstroForth, calculates sky position of several objects from different earth positions. Demos only.
★ **IBM, 700 Kb**

Forth List Handler, Martin Tracy C008 - $8
List primitives extend Forth to provide a flexible, high-speed environment for AI. Incl. ELISA and Winston & Horn's micro-LISP as examples. Incl. source & docs.
★★ **IBM, 170 Kb**

8051 Embedded Forth, William Payne C050 - $20
8051 ROMmable Forth operating system. 8086-to-8051 target compiler. Incl. source. Docs are in the book *Embedded Controller Forth for the 8051 Family.* Included with item #216
★★★ **IBM HD, 4.3 Mb**

68HC11 Collection C060 - $16
Collection of Forths, tools and floating point routines for the 68HC11 controller.
★★★ **IBM HD, 2.5 Mb**

F83 V2.01, Mike Perry & Henry Laxen C100 - $20
The newest version, ported to a variety of machines. Editor, assembler, decompiler, metacompiler. Source and shadow screens. Manual available separately (items 217 & 235). Base for other F83 applications.
★ **IBM, 83, 490 Kb**

F-PC V3.6 & **TCOM** 2.5, Tom Zimmer C200 - $30
A full Forth system with pull-down menus, sequential files, editor, forward assembler, metacompiler, floating point. Complete source and help files. Manual for V3.5 available separately (items 350 & 351). Base for other F-PC applications.
★ **IBM HD, 83, 3.5Mb**

F-PC TEACH V3.5, Lessons 0–7 Jack Brown C201 - $8
Forth classroom on disk. First seven lessons on learning Forth, from Jack Brown of B.C. Institute of Technology.
★ **IBM HD, F-PC, 790 Kb**

VP-Planner Float for F-PC, V1.01 Jack Brown C202 - $8
Software floating-point engine behind the VP-Planner spreadsheet. 80-bit (temporary-real) routines with transcendental functions, number I/O support, vectors to support numeric co-processor overlay & user NAN checking.
★★ **IBM, F-PC, 350 Kb**

F-PC Graphics V4.6, Mark Smiley C203 - $10
The latest versions of new graphics routines, including CGA, EGA, and VGA support, with numerous improvements over earlier versions created or supported by Mark Smiley.
★★ **IBM HD, F-PC, 605 Kb**

PocketForth V6.4, Chris Heilman C300 - $12
Smallest complete Forth for the Mac. Access to all Mac functions, events, files, graphics, floating point, macros, create standalone applications and DAs. Based on fig & *Starting Forth.* Incl. source and manual.
★ **MAC, 640 Kb,** System 7.01 Compatible.

Kevo V0.9b6, Antero Taivalsaari C360 - $10
Complete Forth-like object Forth for the Mac. Object-Prototype access to all Mac functions, files, graphics, floating point, macros, create standalone applications. Kernel source included, extensive demo files, manual.
★★★ **MAC, 650 Kb,** System 7.01 Compatible.

Yerkes Forth V3.67 C350 - $20
Complete object-oriented Forth for the Mac. Object access to all Mac functions, files, graphics, floating point, macros, create standalone applications. Incl. source, tutorial, assembler & manual.
★★ **MAC, 2.4Mb,** System 7.1 Compatible.

Pygmy V1.4, Frank Sergeant C500 - $20
A lean, fast Forth with full source code. Incl. full-screen editor, assembler and metacompiler. Up to 15 files open at a time.
★★ **IBM, 320 Kb**

KForth, Guy Kelly C600 - $20
A full Forth system with windows, mouse, drawing and modem packages. Incl. source & docs.
★★ **IBM, 83, 2.5 Mb**

Mops V2.6, Michael Hore C710 - $20
Close cousin to Yerkes and Neon. Very fast, compiles subroutine-threaded & native code. Object oriented. Uses F-P co-processor if present. Full access to Mac toolbox & system. Supports System 7 (e.g., AppleEvents). Incl. assembler, manual & source.
★★ **MAC, 3 Mb,** System 7.1 Compatible

BBL & Abundance, Roedy Green C800 - $30
BBL public-domain, 32-bit Forth with extensive support of DOS, meticulously optimized for execution speed. Abundance is a public-domain database language written in BBL. Incl. source & docs.
★★★ **IBM HD, 13.8 Mb, hard disk required**

Version-Replacement Policy

Return the old version with the FIG labels and get a new version replacement for 1/2 the current version price.

MISCELLANEOUS

T-SHIRT "May the Forth Be With You" 601 - $12
(Specify size: Small, Medium, Large, X-Large on order form)
white design on a dark blue shirt or green design on tan shirt.

POSTER (*Oct., 1980 BYTE* cover) **Last 10** 602 - $5

ANS-FORTH QUICK REFERENCE CARD 685 - $1

BIBLIOGRAPHY OF FORTH REFERENCES 340 - $18
(3rd ed., January 1987)
Over 1900 references to Forth articles throughout computer literature. *104pgs*

MORE ON FORTH ENGINES

Volume 10 January 1989 810 - $15
RTX reprints from 1988 Rochester Forth conference, object-oriented cmForth, lesser Forth engines. *87 pgs*

Volume 11 July 1989 811 - $15
RTX supplement to *Footsteps in an Empty Valley*, SC32, 32-bit Forth engine, RTX interrupts utility. *93 pgs*

Volume 12 April 1990 812 - $15
ShBoom Chip architecture and instructions, neural computing module NCM3232, pigForth, binary radix sort on 80286, 68010, and RTX2000. *87 pgs*

Volume 13 October 1990 813 - $15
PALs of the RTX2000 Mini-BEE, EBForth, AZForth, RTX2101, 8086 eForth, 8051 eForth. *107 pgs*

Volume 14 814 - $15
RTX Pocket-Scope, eForth for muP20, ShBoom, eForth for CP/M & Z80, XMODEM for eForth. *116 pgs*

Volume 15 815 - $15
Moore: new CAD system for chip design, a portrait of the P20; Rible: QS1 Forth processor, QS2, RISCing it all; P20 eForth software simulator/debugger. *94 pgs*

Volume 16 816 - $15
OK-CAD System, MuP20, eForth system words, 386 eForth, 80386 protected mode operation, FRP 1600 - 16-Bit real time processor. *104 pgs*

Volume 17 817 - $15
P21 chip and specifications; Pic17C42; eForth for 68HC11, 8051, Transputer *128 pgs*

Volume 18 818 - $20
MuP21 - programming, demos, eForth *114 pgs*

Volume 19 819 - $20
More MuP21 - programming, demos, eForth *135pgs*

DR. DOBB'S JOURNAL back issues

Annual Forth issue, includes code for various Forth applications.

Sept. 1982, Sept. 1983, Sept. 1984 (3 issues) 425 - $10

FORTH INTEREST GROUP

P.O. BOX 2154 OAKLAND, CALIFORNIA 94621 510-89-FORTH (510-893-6784) 510-535-1295 (FAX)

Name _____ Phone _____
Company _____ Fax _____
Street _____ eMail _____
City _____ _____
State/Prov. _____ Zip _____
Country _____

Shipping & Handling based on Sub-Total		
U.S. & International Surface	Up to $40.00	$7.50
	$40.01 to $80.00	$10.00
	$80.01 to $150.00	$15.00
	Above $150.00	10% of Total
International Air		40% of Total
Courier Shipments		$15.00 + courier costs

Non-Post Office deliveries: Include Special Instructions.

Item #	Title	Qty.	Unit Price	Total

☐ CHECK ENCLOSED (Payable to: FIG)
☐ VISA/MasterCard Expiration Date _____
Card Number _____
Signature _____

Sub-Total	
10% Member Discount, Member # _____	( )
*Sales Tax on Sub-Total (CA only)	
**Shipping and Handling (see above)	
***Membership in the Forth Interest Group ☐New ☐Renewal $45/53/60	
MEMBERSHIP Total	

New shipping & handling instructions

***MEMBERSHIP IN THE FORTH INTEREST GROUP

The Forth Interest Group (FIG) is a worldwide, nonprofit, member-supported organization with over 1,000 members and 10 chapters. Your membership includes a subscription to the bimonthly magazine *Forth Dimensions*. FIG also offers its members an on-line data base, a large selection of Forth literature and other services. Cost is $45 per year for U.S.A. & Canada surface; $53 Canada air mail; all other countries $60 per year. This fee includes $39 for *Forth Dimensions*. No sales tax, handling fee, or discount on membership.
When you join, your first issue will arrive in four to six weeks; subsequent issues will be mailed to you every other month as they are published—six issues in all. Your membership entitles you to a 10% discount on publications and functions of FIG. Dues are not deductible as a charitable contribution for U.S. federal income tax purposes, but may be deductible as a business expense.

MAIL ORDERS:
Forth Interest Group
P.O. Box 2154
Oakland, CA 94621
PHONE ORDERS:
510-89-FORTH Credit card orders, customer service. Hours: Mon–Fri, 9–5 p.m.

PAYMENT MUST ACCOMPANY ALL ORDERS

PRICES: All orders must be prepaid. Prices are subject to change without notice. Credit card orders will be sent and billed at current prices. Checks must be in U.S. dollars, drawn on a U.S. bank. A $10 charge will be added for returned checks.

SHIPPING & HANDLING:
All orders calculate shipping & handling based on order dollar value.
Special handling available on request.

SHIPPING TIME:
Books in stock are shipped within seven days of receipt of the order.
***SURFACE DELIVERY:**
US: 10 days, Other: 30-60 days

*** CALIFORNIA SALES TAX BY COUNTY:**
7.75%: Del Norte, Fresno, Imperial, Inyo, Madera, Orange, Riverside, Sacramento, Santa Clara, Santa Barbara, San Bernardino, San Diego, and San Joaquin; **8.25%**: Alameda, Contra Costa, Los Angeles San Mateo, San Francisco, San Benito, and Santa Cruz; **7.25%**: other counties.

For faster service, fax your orders: 510-535-1295

Listing

```
( DFC - Differential File Comparison.  Wil Baden 1976-1996 )

( Make a line by line comparison of two files, showing
( where and how they are different. )

( Used:
    S" <oldfilename>" INPUT TO OLD
    S" <newfilename>" INPUT TO NEW
    DFC
)

: BOUNDS    OVER + SWAP ;        ( a k -- a+k a )

: INPUT     R/O OPEN-FILE ABORT" Can't open " ;

: REWIND                         ( fileid -- )
            0 0 ROT REPOSITION-FILE
                ABORT" Sorry, error rewinding file. "
;

: PLACE                          ( s . a -- )
            2DUP >R >R  CHAR+  SWAP CHARS MOVE  R> R> C!
;

: UNDER     ROT DROP SWAP ;      ( a b c -- c b )

: hash                   ( counted-string -- hash-value)
    ( Compute hash value for a counted string. )
    0 SWAP COUNT CHARS BOUNDS ?DO    ( hash-value)
        65599 *   I C@ +
    1 CHARS +LOOP
;

0   VALUE       OLD
0   VALUE       NEW

254 CONSTANT    linesize

CREATE      oldtext     linesize 3 + CHARS ALLOT
CREATE      newtext     linesize 3 + CHARS ALLOT

: read-text                      ( buffer fileid -- flag )
    >R CHAR+ DUP   linesize  R> READ-LINE
        ABORT" Sorry, error reading file. "
    ROT ROT -TRAILING SWAP 1 CHARS - C!
;

( Cell for each record + 3*matching-candidates. )
( Thus 6000 handles files up to 1200 lines or more. )

6000 CONSTANT   lcs-space       ( The larger the better. )
CREATE          LCS   lcs-space CELLS ALLOT

0 VALUE oldlines        0 VALUE newlines
VARIABLE X              VARIABLE Y
VARIABLE cand
```

(Continues.)

```
( newlines : 1 + lines in newer file. )
( oldlines : 1 + lines in old file + 1 + lines in new file. )
( cand : next candidate. )
( LCS : in Find-LCS, pointer to candidate;)
(     in Show-Diffs, number of matched lines. )
( X Y : generally, working variables;
(     in Show-Diffs, old-line-number and new-line-number. )

: slot     CELLS LCS + ;                    ( i -- a )
: slot-h   lcs-space SWAP - CELLS LCS + ;   ( i -- a )

( Read in the newer file, which is generally longer. Work
( from both ends toward the middle.  From the beginning of
( LCS put in the line numbers: 1, 2, 3, ....  From the end
( of LCS put in corresponding hash values: ... h3, h2, h1.
( Cell LCS[0] is not used. )

: read-newerfile                        ( -- )
    ( Read newer file saving line numbers and hash values. )
    ( Output: newlines ; Use: newtext )
    0                                   ( n)
    BEGIN
        1+
        newtext NEW read-text
    WHILE
        DUP 2* lcs-space > ABORT" Sorry, not enough space. "
        DUP  DUP slot !
        newtext hash OVER slot-h !
    REPEAT
    TO newlines                         ( )
;

( Order the hash values, carrying the line numbers as the
( minor key. The result has the first n-1 line numbers in
( the cells 1..n-1 sorted by the hash values of the
( corresponding line. )

: insert-hash-value                 ( Gap j -- Gap )
    ( Inner insertion loop for custom Shell sort. )
    ( Use: X Y )
    DUP slot-h @  X !    DUP slot @  Y !
    OVER -                              ( Gap j)
    BEGIN   DUP slot-h @  X @ < NOT
    WHILE
        DUP slot-h @  X @ >
        ?DUP 0= IF   DUP slot @  Y @ >  THEN
    WHILE
        2DUP + >R
            DUP slot-h @    R@ slot-h !
            DUP slot @
        R> slot !
        OVER -
        DUP 1 <
    UNTIL   THEN THEN   OVER +          ( Gap j+Gap)
    X @  OVER slot-h !    Y @  OVER slot !
    DROP                                ( Gap)
;
```

```
: sort-hash-values                    ( -- )
    ( Shell sort for unusual data structure. )
    ( Input: newlines )
    newlines   1                          ( lines gap)
    BEGIN   2DUP 1+ > WHILE  2*  1+  REPEAT
    BEGIN   2/   DUP
    WHILE
        2DUP 1+ DO   I insert-hash-value   LOOP
    REPEAT                                2DROP
;

( Mark the hash value equivalence classes by negating
( the last line number associated with a hash value. )

: mark-hash-classes                   ( -- )
    ( Negate lines with different hash from next line. )
    ( Input: newlines )
    newlines 1-  1  DO
        I slot-h @  I 1+ slot-h @ = NOT
            IF   I slot DUP @ NEGATE SWAP !   THEN
    LOOP
    newlines 1- slot DUP @ NEGATE SWAP !
;

( Reserve two cells following the line numbers of the newer
( file. Now read in each line of the old file. Take the hash
( value of the line, and find the first line in the newer
( file having the same hash value. Store the number of the
( cell containing line number in the next successive cell.
( If the line in the old file does not appear anywhere in
( newer file, store 0. )

    : search-for-hash   ( match high low hash -- match )
        >R                        ( match high low)( R: hash)
            BEGIN   OVER 1+ OVER <
            WHILE
                2DUP + 2/             ( match low high mid)
                DUP slot-h @  R@ < IF
                    UNDER             ( match low high)
                ELSE                  ( match low high mid)
                    NIP               ( match low high)
                    DUP slot-h @  R@ =
                        IF   UNDER OVER   THEN
                THEN
            REPEAT   2DROP            ( match)
        R> DROP                       ( R: )
    ;

: read-oldfile                        ( -- )
    ( Read oldfile and match newfile hashed lines. )
    ( Input: newlines ; Output: oldlines )
    newlines 1+                       ( biased-line-number)
    BEGIN
        1+
        oldtext OLD read-text
    WHILE
        DUP newlines + lcs-space >
            ABORT" Sorry, out of space for newer file. "
```

(Continues.)

```
174         0 0 newlines                      ( . match low high)
175         oldtext hash search-for-hash    ( biased-line match)
176         OVER slot !                     ( biased-line-number)
177     REPEAT
178     TO oldlines                         ( )
179 ;

181 ( We are done with the sub-array of hash values, and the
182 ( memory can be used for something else. )

184 ( Find the longest common subsequence.  Following the
185 ( sub-array used for the old file, build a doubly-linked
186 ( list representing the potential longest common subsequences
187 ( in reverse order. In doing this, replace the value in
188 ( the cells associated with the old file with the cell number
189 ( of the appropriate doubly-linked list. The two cells that
190 ( were reserved are used as the bounds of the subsequences. )

192 : candidate                  ( x y z -- candidate-pointer)
193     ( Make a new candidate for LCS. )
194     ( In/Out: cand )
195     cand @ lcs-space 2 - >
196         ABORT" Sorry, candidate space exhausted. "
197     cand @ >R                         ( R: candidate-pointer)
198         >R >R                         ( x)
199             cand @ slot !             (  )
200             1 cand +!
201             R> ( y) cand @ slot !     (  )
202             1 cand +!
203         R> ( z) cand @ slot !         (  )
204         1 cand +!
205     R>                            ( candidate-pointer)( R: )
206 ;

208 : search-for-match ( Value low high -- 0 | Value wherefound )
209     ( Binary search for LCS candidates. )
210     ROT >R                            ( low high)( R: Value)
211         BEGIN   2DUP > NOT
212         WHILE
213             2DUP + 2/                 ( low high mid)
214             DUP slot @ 1+ slot @  R@ < NOT IF
215                 1-    NIP             ( low high)
216             ELSE                      ( low high mid)
217                 DUP 1+ slot @ 1+ slot @  R@ < NOT
218                     IF    NIP NIP R> SWAP EXIT    THEN
219                 1+    UNDER           ( low high)
220             THEN
221         REPEAT                            2DROP
222     R> DROP                           ( R: )
223     0                                 ( 0)
224 ;

226 : new-candidate          ( value wherefound i -- flag)
227     ( Make and link a new LCS candidate. )
228     ( In/Out: X Y LCS )
229     ROT ROT                       ( i value wherefound)
230     DUP >R
231         2DUP 1+ slot @ 1+ slot @  < IF
```

```
232             Y @  X @ slot !
233             DUP 1+  X !
234             slot @  candidate Y !          ( )
235         ELSE   2DROP DROP   THEN
236     R>  LCS @ =                            ( flag)
237     DUP IF ( Move fence. )
238         LCS @ 1+ slot @  LCS @ 2 + slot !
239         1 LCS +!
240     THEN                                   ( flag)
241 ;

243 : find-longest-common-subsequence   ( -- )
244     ( Nuf ced. )
245     ( Input: oldlines newlines ; Use: cand LCS X Y )
246     oldlines  cand !
247     newlines  LCS !
248     newlines 1+  0 0 candidate  LCS @ slot !
249     oldlines  newlines 0 candidate  LCS @ 1+ slot !
250     oldlines  newlines 2 +
251     DO
252         I slot @                  ( newer-line-number)
253         DUP IF
254             newlines  DUP X !  slot @  Y !
255             BEGIN
256                 DUP slot @ ABS  ( . value)
257                 X @  LCS @  search-for-match
258                                   ( . 0 | . value wherefound)
259                 DUP IF   I  new-candidate   THEN
260                                   ( newer-line-number flag)
261                 0=
262             WHILE                 ( newer-line-number)
263                 DUP slot @ 0>
264             WHILE
265                 1+
266             REPEAT THEN
267             Y @  X @ slot !
268         THEN                      DROP
269     LOOP
270 ;

272 ( Untangle the linked reverse list of the longest common
273 ( subsequence to become a simple linear list in forward
274 ( order in the sub-array used for the old file. )

276 : build-candidate-table             ( -- )
277     ( Unravel LCS. )
278     ( Input: LCS oldlines newlines )
279     LCS @ slot @                          ( c)
280     oldlines  newlines 2 +
281         DO    0  I slot !    LOOP
282     newlines  oldlines  slot !
283     BEGIN   DUP
284     WHILE
285         DUP 1+ slot @                     ( c j)
286         OVER slot @ slot !                ( c)
287         2 + slot @
288     REPEAT                                DROP
289 ;
```

(Continues.)

```
( The values are 0 if the line does not appear in the newer
( file, or the line number of a candidate match in the
( newer file.  Skipped numbers are lines that are new in
( the newer file. )

( Display the lines that were deleted from the old file,
( inserted in the newer file, or unchanged. )

( State: 0= delete, 0< add, 0> copy. )

VARIABLE matchingtext   linesize 3 + CHARS ALLOT

: deleted                    ( previous-state -- state )
    ( What to do when the line is in the old file only. )
    ( Input: X Y oldtext )
    ( In/Out: matchingtext )
    matchingtext C@ IF
        X @ 1- 4 U.R SPACE
            Y @ 4 U.R    SPACE
                matchingtext COUNT 1- TYPE CR
        0 matchingtext C!
    THEN
    X @ 4 U.R    SPACE    ." DEL> "
        oldtext COUNT TYPE CR
    DROP 0 ( delete )
;

: added                      ( previous-state -- state )
    ( What to do when the line is in the newer file only. )
    ( Input: X Y newtext )
    ( In/Out: matchingtext )
    matchingtext C@ IF
        X @ 1- 4 U.R    SPACE
            Y @ 1- 4 U.R   SPACE
                matchingtext COUNT 1- TYPE CR
        0 matchingtext C!
    THEN
    ." NEW> "    Y @ 4 U.R    SPACE
        newtext COUNT TYPE CR
    DROP -1 ( add )
;

: matched                    ( previous-state -- state )
    ( What to do when the line is in both files. )
    ( Input: X Y oldtext newtext )
    ( In/Out: LCS : number of matched lines. )
    ( Output: matchingtext )
    1 LCS +!
    DUP 1- 0< ( adding or deleting ) IF
        X @ 4 U.R    SPACE
            Y @ 4 U.R    SPACE
                newtext COUNT TYPE CR
        DROP 1 ( copy )
    ELSE ( copying, = number of lines just copied. )
        1+
        3 OVER = IF    CR    THEN
        newtext COUNT 1+ matchingtext PLACE
```

```
348     THEN
349 ;

351     : handle-deleted            ( state -- same )
352         BEGIN
353             1 X +!    X @ newlines + 1+  oldlines  < IF
354                 oldtext OLD read-text 0=
355                     ABORT" Oops, error with old file. "
356             THEN
357             X @ newlines + 1+  slot @
358                 ( i.e. newer-line-number) 0=
359         WHILE   deleted   REPEAT
360     ;

362     : handle-added              ( state -- same )
363         BEGIN
364             1 Y +!    Y @  newlines < IF
365                 newtext NEW read-text 0=
366                     ABORT" Oops, error with newer file. "
367             THEN
368             X @ newlines + 1+  slot @ Y @ >
369         WHILE   added   REPEAT
370     ;

372     : handle-matched            ( state -- same )
373     ( Check that matched records are really the same. )
374         oldtext COUNT newtext COUNT COMPARE 0= IF
375             matched
376         ELSE   added   deleted   THEN
377     ;

379 : show-differences               ( -- )
380     ( Let's see them. )
381     ( Input: oldlines newlines ; Use: X Y LCS matchingtext )
382     OLD REWIND   NEW REWIND
383     0 X !    0 Y !    0 LCS !
384     0 matchingtext C!
385     1 ( copying )
386     BEGIN                              ( state)
387         handle-deleted    handle-added
388         Y @ newlines <
389     WHILE   handle-matched
390     REPEAT                            DROP
391 ;

393 : DFC                              ( -- )
394     ( Differential file comparison. )
395     read-newerfile   sort-hash-values   mark-hash-classes
396     read-oldfile     find-longest-common-subsequence
397     build-candidate-table   show-differences
398     OLD REWIND   NEW REWIND
399 ;

401 \ Procedamus in pace.    Wil Baden    Costa Mesa, California
```

Using Forth to manipulate the real world

Forthware

Getting to the Hardware from Linux

Skip Carter
Monterey, California

1995 was a year in which probably everybody using a PC agonized over the prospect of changing operating systems. For myself, I thought about it and ultimately installed Linux on my PC. I have to confess to a Unix prejudice (brought about by nearly two decades of exposure to it!) so I felt right at home. Those of you who moved to Linux without any previous experience with minicomputers and workstations were probably shocked to discover one fact about sophisticated operating systems: you no longer control the machine, the operating system does and you have to ask permission from it to do what used to be an ordinary thing (such as writing to the parallel port).

In this month's column, we will give you Linux newcomers the ability to catch up with all those MS-DOS users who are terrorizing their cats with stepper-motor-controlled "mice."

Which Forth to Use

If you are running with the Linux operating system you have three choices when it comes to Forth compilers,

- The standard write-it-yourself approach.
- Wil Baden's ThisForth.
- Dirk Uwe Zoller's PFE (Portable Forth Environment).

The second two are good solid compilers, good enough that the only justification for doing your own is the educational value. Wil's compiler is the most portable Forth-in-C implementation I have ever seen (I even got it running on a Cray, a 64-bit environment). It is not quite ANS (it was never intended for that purpose), but one can write ANS programs on it. It requires implementing the ANS File wordset on top of the (more generalized) I/O system that ThisForth implements. PFE is a very good ANS Forth system; it is not as portable as ThisForth, but it runs much more quickly. PFE running on Linux also has the very nice feature of the ability to naturally access shared libraries (these are the DLLs of the Unix/Linux world).

Dirk has also managed to do the Forth community a service by getting PFE as an installation option of Linux distribution CD-ROMS (also, by the way, the FreeBSD operating system distribution CD-ROM).

Where did PRN go?

The first thing to learn is where the parallel port went to. Under Linux, one can't just go read from address 0040:0008. First of all, Linux presents all processes with a flat virtual memory space, it takes a special system call to convert to the actual physical location. Second, that physical memory location is only significant to MS-DOS. Instead, Linux does has a table of all the physical and virtual devices that the operating system knows about. This table is the special directory, /dev. The Linux devices /dev/parX are the parallel ports, the value of X depends upon the address of the port:

Table One. Correspondence of devices and addresses.

Device	Address	Major number	Minor number
/dev/par0	0x3BC	6	0
/dev/par1	0x378	6	1
/dev/par2	0x278	6	2

The *major device* number is an index into a list of device drivers; on my system, index 6 is the parallel port driver (the value of the number is arbitrary but needs to be established uniquely for each device when building the operating system image. The Linux community has adopted a major device number convention for standard devices to make interoperability and management simpler. Other than following convention, there is no reason why the value of this number is special).

The *minor device* number is an index used internally by the device driver. It is typically used to indicate an instance of the actual device; in the above case, the system is capable of handling three different parallel ports. They do not have to all actually exist, on my system the only one that actually corresponds to a physical port is /dev/par1. If you have the standard parallel port device driver compiled into the operating system, then you can figure out which one is your actual port by looking up the port address in Table One.

You can find out the assignment of the major and minor

Figure One. Major and minor device number assignments.

```
crw-rw----   1 root     daemon     6,   0 Jul 17  1994 /dev/par0
crw-rw-rw-   1 root     daemon     6,   1 Jul 17  1994 /dev/par1
crw-rw----   1 root     daemon     6,   2 Jul 17  1994 /dev/par2
```

device numbers to all the devices by typing the command `ls -l /dev/par*` *[see Figure One]*.

Going from left to right this tells us,

- The permission settings (see below).
- The number of hard links (we will ignore this here).
- The name of the owner, `root` is the master of the system.
- The name of the group that this device belongs to, `daemon` is the group of programs that run in the background performing system services.
- The major device number, 6 in all cases for this device.
- The minor device number, 0 through 2.
- The creation date.
- The name of the device driver.

Getting to the Port

Now that we know where the port is, how do we talk to it? The short answer is, you treat /dev/parX as if it were a disk file and just open the file and write to it. The long answer deals with all the complicating factors that makes it actually work.

First, you need to have permission to open the file. Looking at the directory listing shown above we see, `crw-rw----`. The `c` means we are dealing with a character device. The first two `rw`'s mean that the owner and group have read/write privilege on the device and everyone else (including you, most likely) can't touch.

We need to get permission to use the device. There are two solutions to this problem: (i) become the privileged owner or part of the privileged group, or (ii) expand the read/write privileges on the device. The first is not a good idea because that means that you need to become the super-user (`root`) to run an application; this can be dangerous, since `root` has the privilege to do almost *anything*. A slightly safer method is to make the permissions on your program such that it runs with the super-user privilege level even though you are not; this is done by setting the setuid bit with the `chmod` command. It is much safer to extend the permissions on the device to everybody by using the command `chmod +rw` on the device (of course `root` has to do this).

I did this on my system, which is why the permissions on /dev/par1 looks the way it does.

Now we can open the device with the (ANS) Forth line:

```
S" /dev/par1" R/W BIN OPEN-FILE
ABORT" Unable to open parallel port at
/dev/par1"
```

The use of `BIN` to modify the file access method is not really necessary, but it is probably a good habit to use when manipulating bits on a file or device. Read access is also probably not that useful either, unless you have bi-directional parallel port hardware.

Why test for an open failure when we have fixed the permissions on the device? Remember that Linux is a multi-user/multi-tasking operating system, lots of things are going on (even when nobody is logged in). It could be that some other user or process has opened the device. If this happens, your attempt to open the device will fail until the device is unclaimed again.

Talking to the Port

Once you have the permissions all properly set up, you can get to the port. Listing One, fcontrol.fth is a Linux version of the MS-DOS code originally presented by Ken Merk (*FD* XVII/2). If you are using the standard device driver for the parallel port, you will find that you can only write, at most, one character to the port, then your program hangs. This is because the device driver expects the BUSY pin (pin 11) to be pulled low when the port is ready for another character. The easiest way to handle this is to wire your cable connector to loop back one of the ground pins (18 to 25) to pin 11. This way your device always looks ready.

Controlling the Stepper Motors

At this point, we can actually use the parallel port for our own needs. By using our new version of fcontrol.fth, we can drive stepper motors by using the file steppers.seq from last time by making the following simple change: replace the lines,

```
fload fpc2ans.seq
fload fsl-util.seq
fload structs.seq
fload fcontrol.seq
```

with the lines,

```
s" fsl-util.fth" included
s" structs.seq"  included
s" fcontrol.fth" included
```

Other Devices

All this is just fine if you just want to manipulate the Data bits of the port, but you can't do much with the Command and Status bits this way. To do this, you go back to the problem of running your program with root privilege and calling the I/O port directly (with the dangers this implies), or you write a special device driver. The second approach is a vastly better method if you are going to be making a regular practice of using the port; it is also what you should do if you just got your hands on a fancy new A/D board that you want to use.

A Linux/Unix device driver is nothing more than an

I/O interface with a standard set of entry points (primarily `open`, `close`, `read`, `write`, and a configure routine). The only trick is that the device drivers are *statically* bound into the operating system; this means that you have to rebuild the operating system kernel to add a new device driver. This is one reason that you will see lots of stuff in /dev that are not actually associated with any installed device. This allows you to, say, add a CD-ROM drive to your system and then just use it without having to recompile the operating system in order to get to it.

Feedback

Recall that last time I said that the coils in the stepper motor act as inductors storing current. Dwight Elvey sent me e-mail pointing out that this means that one of the limiting factors on how fast you can drive the stepper motor is how long it takes to dump the coil current (if you drive it too fast, barely any current is gone before you have gone and recharged it, so the motor will never see the step—inductors are used, after all, as a component in low-pass filters). Dwight describes that he has used a couple of techniques to cause the coils to discharge their current more quickly. The most reliable method is to put a zener diode in series with the shunting diodes. He reports that he has increased the useful step rate by as much as a factor of two by using this technique.

Conclusion

Now that we have wrestled control of our computer back into our hands, we can join everybody else and use the machines to control the world. We will start by controlling electrical power, which is the topic of our next column.

Please send your comments, suggestions, and criticisms to me through *Forth Dimensions* or via e-mail at skip@taygeta.com.

Skip Carter is a scientific and software consultant. He is the leader of the Forth Scientific Library project, and maintains the system taygeta on the Internet. He is also the President of the Forth Interest Group. The code that accompanies this article is on-line at ftp://ftp.forth.org/pub/Forth/FD/1996 for downloading.

ADVERTISERS INDEX

Listing One. fcontrol.fth

```
\ fcontrol.fth          Forth code to control parallel printer port.
\                       See Ken Merk, Forth Dimensions, July 1995.

\  Converted for PFE under Linux              EFC   October 1995

\ This is an ANS Forth program requiring:
\         1. The File Access word set
\         2. The word FLUSH-FILE from the File Access Extensions
\            word set

\  Note: in order to use this code
\        1. Linux should be using the standard polled printer driver
\        2. DB-25 Pin 11 (BUSY) should be tied low, this can be easily
\           achieved by looping pin 11 back to one of pins 18-25
\        3. The proper /dev/parX device is set in INIT-PORT below
\        4. The permissions on that device should be:   crw-rw-rw-
\ ================================================================

CR .(  FCONTROL.FTH                    October 1995  )
: FCONTROL.FTH ;

\ the Linux devices /dev/parX are the parallel ports,
\ the value of X depends upon the address of the port:
\          X       Address
\          0       0x3BC
\          1       0x378
\          2       0x278

: init-port ( -- n )
         S" /dev/par1" R/W BIN OPEN-FILE
        ABORT" Unable to open parallel port at /dev/par1"
;
init-port CONSTANT #PORT

CREATE cbuf    8 ALLOT

: pc!  ( n port -- )
          DUP ROT cbuf C! cbuf 1 ROT  WRITE-FILE DROP
        FLUSH-FILE DROP
;

: pc@  ( port -- n )
          cbuf 1 ROT READ-FILE 2DROP cbuf C@
;

HEX
: KILL ( -- )        00 #PORT pc!  ;     \ turn OFF all devices
: ALL-ON ( -- )    0FF #PORT pc!  ;     \ turn ON all devices

: ON?    ( b -- f ) #PORT pc@ AND 0<> ;  \ get ON status of device
: OFF?   ( b -- f ) #PORT pc@ AND 0=  ;  \ get OFF status of device

: WRITE ( b -- )    #PORT pc! ;          \ WRITE byte to port
: READ  ( -- b )    #PORT pc@ ;          \ READ byte at port
DECIMAL

\ a crude wait approximately n milliseconds
\ adjust the "700" value for your system

\ : ms  ( n -- )       0 DO   700 0 DO LOOP  LOOP ;
```

Listing Two. fsl-util.fth

```
\ fsl-util.fth          An auxiliary file for the Forth Scientific Library
\                       For PFE

\                       contains commonly needed definitions.
\ dxor, dor, dand       double xor, or, and
\ sd*                   single * double = double_product
\ v: defines use( &     For defining and settting execution vectors
\ %                     Parse next token as a FLOAT
\ S>F  F>S              Conversion between (single) integer and float
\ F,                    Store FLOAT at (aligned) HERE
\ F=                    Test for floating point equality
\ -FROT                 Reverse the effect of FROT
\ F2*  F2/              Multiply and divide float by two
\ F2DUP                 FDUP two floats
\ F2DROP                FDROP two floats
\ INTEGER, DOUBLE, FLOAT   For setting up ARRAY types
\ ARRAY DARRAY              For declaring static and dynamic arrays
\ }                         For getting an ARRAY or DARRAY element address
\ &!                        For storing ARRAY aliases in a DARRAY
\ PRINT-WIDTH               The number of elements per line for printing arrays
\ }FPRINT                   Print out a given array
\ Matrix                    For declaring a 2-D array
\ }}                        gets a Matrix element address
\ Public: Private: Reset_Search_Order   controls the visibility of words
\ frame unframe             sets up/removes a local variable frame
\ a b c d e f g h           local FVARIABLE values
\ &a &b &c &d &e &f &g &h   local FVARIABLE addresses

\ This code conforms with ANS requiring:
\       1. The Floating-Point word set
\       2. The words umd* umd/mod and d* are implemented
\          for ThisForth in the file umd.fo

\ This code is released to the public domain Everett Carter July 1994

CR .( FSL-UTIL.FTH      V1.15           7 October    1995   EFC )

\ ================ compilation control ====================================

\ for control of conditional compilation of test code
FALSE VALUE TEST-CODE?
FALSE VALUE ?TEST-CODE          \ obsolete, for backward compatibility

\ for control of conditional compilation of Dynamic memory
TRUE CONSTANT HAS-MEMORY-WORDS?

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

\ FSL NonANS words

: ~DEFINED   ( c-addr -- t/f )    \ returns definition status of
      FIND SWAP DROP 0=           \ a word
;

WORDLIST CONSTANT hidden-wordlist

: Reset-Search-Order
      FORTH-WORDLIST 1 SET-ORDER
      FORTH-WORDLIST SET-CURRENT
;
```

```
: Public:
     FORTH-WORDLIST hidden-wordlist 2 SET-ORDER
     FORTH-WORDLIST SET-CURRENT
;

: Private:
     FORTH-WORDLIST hidden-wordlist 2 SET-ORDER
     hidden-wordlist SET-CURRENT
;

: Reset_Search_Order   Reset-Search-Order ;      \ these are

CREATE fsl-pad    84 CHARS ( or more ) ALLOT

: dxor       ( d1 d2 -- d )              \ double xor
     ROT XOR >R XOR R>
;

: dor        ( d1 d2 -- d )              \ double or
     ROT OR >R OR R>
;

: dand      ( d1 d2 -- d )               \ double and
     ROT AND >R AND R>
;

: d>
        2SWAP D<
;

\ single * double = double
: sd*   ( multiplicand  multiplier_double  -- product_double  )
            2 PICK * >R   UM*   R> +
;

: CELL-     [ 1 CELLS ] LITERAL - ;           \ backup one cell

0 VALUE TYPE-ID                \ for building structures
FALSE VALUE STRUCT-ARRAY?

\ for dynamically allocating a structure or array

TRUE  VALUE is-static?      \ TRUE for statically allocated structs and arrays
: dynamic ( -- )      FALSE TO is-static? ;

\ size of a regular integer
1 CELLS CONSTANT INTEGER

\ size of a double integer
2 CELLS CONSTANT DOUBLE

\ size of a regular float
1 FLOATS CONSTANT FLOAT

\ size of a pointer (for readability)
1 CELLS CONSTANT POINTER

: % BL WORD COUNT >FLOAT 0= ABORT" NAN"
                  STATE @ IF POSTPONE FLITERAL THEN ; IMMEDIATE
```

(Continues.)

```
: F,   HERE FALIGN 1 FLOATS ALLOT F! ;

\ 1-D array definition
\    ----------------------------
\    | cell_size | data area     |
\    ----------------------------

: MARRAY ( n cell_size -- | -- addr )              \ monotype array
    CREATE
      DUP , * ALLOT
    DOES> CELL+
;

\    ----------------------------
\    | id | cell_size | data area |
\    ----------------------------

: SARRAY ( n cell_size -- | -- id addr )           \ structure array
    CREATE
      TYPE-ID ,
      DUP , * ALLOT
    DOES> DUP @ SWAP [ 2 CELLS ] LITERAL +
;

: ARRAY
    STRUCT-ARRAY? IF   SARRAY FALSE TO STRUCT-ARRAY?
                  ELSE MARRAY
                  THEN
;

\ word for creation of a dynamic array (no memory allocated)

\ Monotype
\    ------------------------
\    | data_ptr | cell_size |
\    ------------------------

: DMARRAY   ( cell_size -- )   CREATE  0 , ,
                               DOES>
                                     @ CELL+
;

\ Structures
\    ----------------------------
\    | data_ptr | cell_size | id |
\    ----------------------------

: DSARRAY   ( cell_size -- )  CREATE  0 , , TYPE-ID ,
                              DOES>
                                    DUP [ 2 CELLS ] LITERAL + @ SWAP
                                     @ CELL+
;

: DARRAY   ( cell_size -- )
    STRUCT-ARRAY? IF   DSARRAY FALSE TO STRUCT-ARRAY?
                  ELSE DMARRAY
                  THEN
;
```

```
\ word for aliasing arrays,
\  typical usage:  a{ & b{ &!  sets b{ to point to a{'s data

: &!     ( addr_a &b -- )
         SWAP CELL- SWAP >BODY  !
;

: }    ( addr n -- addr[n])        \ word that fetches 1-D array addresses
          OVER CELL-  @
          * SWAP +
;

VARIABLE print-width       6 print-width !

: }fprint ( n addr -- )        \ print n elements of a float array
         SWAP 0 DO I print-width @ MOD 0= I AND IF CR THEN
                   DUP I } F@ F. LOOP
         DROP
;

: }iprint ( n addr -- )        \ print n elements of an integer array
         SWAP 0 DO I print-width @ MOD 0= I AND IF CR THEN
                   DUP I } @ . LOOP
         DROP
;

: }fcopy ( 'src 'dest n -- )          \ copy one array into another
     0 DO
                OVER I } F@
                DUP  I } F!
          LOOP
          2DROP
;

\ 2-D array definition,

\ Monotype
\    -----------------------------
\    | m | cell_size |  data area |
\    -----------------------------

: MMATRIX  ( n m size -- )            \ defining word for a 2-d matrix
         CREATE
            OVER , DUP ,
            * * ALLOT
         DOES>  [ 2 CELLS ] LITERAL +
;

\ Structures
\    ---------------------------------
\    | id | m | cell_size |  data area |
\    ---------------------------------

: SMATRIX  ( n m size -- )            \ defining word for a 2-d matrix
         CREATE TYPE-ID ,
            OVER , DUP ,
            * * ALLOT
         DOES>  DUP @ TO TYPE-ID
                [ 3 CELLS ] LITERAL +
;
```

(Continues.)

```
: MATRIX  ( n m size -- )           \ defining word for a 2-d matrix
     STRUCT-ARRAY? IF   SMATRIX FALSE TO STRUCT-ARRAY?
                   ELSE MMATRIX
                   THEN
;

: DMATRIX ( size -- )      DARRAY ;


: }}    ( addr i j -- addr[i][j] )     \ word to fetch 2-D array addresses
              2>R                      \ indices to return stack temporarily
              DUP CELL- CELL- 2@       \ &a[0][0] size m
              R> * R> + *
              +
;

: }}fprint ( n m addr -- )         \ print nXm elements of a float 2-D array
        ROT ROT SWAP 0 DO
                          DUP 0 DO
                                   OVER J I  }} F@ F.
                          LOOP
                          CR
                    LOOP
        2DROP
;


\ function vector definition

: v: CREATE ['] noop , DOES> @ EXECUTE ;
: defines   ' >BODY STATE @ IF POSTPONE LITERAL POSTPONE !
                             ELSE ! THEN ;   IMMEDIATE

: use(  STATE @ IF POSTPONE ['] ELSE ' THEN ;  IMMEDIATE
: &     POSTPONE use( ; IMMEDIATE

(
  Code for local fvariables, loosely based upon Wil Baden's idea presented
  at FORML 1992. The idea is to have a fixed number of variables with fixed names.
  I believe the code shown here will work with any, case insensitive, ANS Forth.

  i/tForth users are advised to use FLOCALS| instead.

  example:  : test  2e 3e FRAME| a b |  a f. b f. |FRAME ;
            test <cr> 3.0000 2.0000 ok

  PS: Don't forget to use |FRAME before an EXIT .
)

8 CONSTANT /flocals

: (frame) ( n -- ) FLOATS ALLOT ;

: FRAME|
        0 >R
        BEGIN   BL WORD  COUNT  1 =
                SWAP C@  [CHAR] | =.
                AND 0=
        WHILE   POSTPONE F,  R> 1+ >R
        REPEAT
```

```
        /FLOCALS R> - DUP 0< ABORT" too many flocals"
        POSTPONE LITERAL  POSTPONE (frame) ; IMMEDIATE

: |FRAME ( -- ) [ /FLOCALS NEGATE ] LITERAL (FRAME) ;

: &h            HERE [ 1 FLOATS ] LITERAL - ;
: &g            HERE [ 2 FLOATS ] LITERAL - ;
: &f            HERE [ 3 FLOATS ] LITERAL - ;
: &e            HERE [ 4 FLOATS ] LITERAL - ;
: &d            HERE [ 5 FLOATS ] LITERAL - ;
: &c            HERE [ 6 FLOATS ] LITERAL - ;
: &b            HERE [ 7 FLOATS ] LITERAL - ;
: &a            HERE [ 8 FLOATS ] LITERAL - ;

: a             &a F@ ;
: b             &b F@ ;
: c             &c F@ ;
: d             &d F@ ;
: e             &e F@ ;
: f             &f F@ ;
: g             &g F@ ;
: h             &h F@ ;
```

(Fast Forthward, from page 43.)

in order to discover the few that are broken. To test each link, you must be able to recognize the intended destination when you reach it. I wish you luck if you ever have to do this for a nontrivial project. Testing a few thousand distinct destinations this way is a hardship you would rather not confront.

First, you must click correctly through as many as four or more levels of heads before you reach or fail to reach an expected book passage. The initial clicks take you to the *intermediate destinations*, corresponding to the drill-down navigation screens. Final destinations are book passages. For a straightforward book conversion such as ours, they will not possess more links of their own.

The mouse clicks can really add up as you seek each book-passage. Suppose you have a book—or suite of books—with a rate of headline fan-out of ten that continues for three levels of heads, followed by a fan-out of four for fourth-level headlines. This would yield a count of about 4,000 ($4^1 * 10^3$) destination passages at header level five. A complete-coverage test in such a scenario involves intermediate-destination checks totaling 1,110 (10 at level two, plus 100 at level three, plus 1,000 at level four), in addition to the 4,000 final-destination checks. That's a minimum of 5,110 mouse clicks and 5,110 companion, where-am-I-now assessments.

New Book-Building Tools

My tools prove that automation need not impose severe limitations nor impact the quality of the on-line books that can be produced.

After a flurry of processing, the newly developed tools will generate and safely store a single `hlp`-suffixed file for each FrameMaker book. Along the way to that final file, about 20 intermediate files will be generated for a book that has five chapter files and five heading levels. Most of these are short-lived files built in a temporary holding area.

Five of the files will provide the drill-down navigation capability that I sought so keenly. (The number of drill-down files generated varies. Roughly one is needed for each level of headline present in the original book.)

These drill-down files start out as what FrameMaker calls a book's *generated* files. They are initially created by FrameMaker as though they were ordinary table-of-contents files. However, each of these files was carefully defined to focus exclusively upon one header level.

These drill-down files are processed by a custom awk program. It redirects the Frame-generated hypertext links that normally always point to the book's body text. To support drill-down navigation, most level-one heads are redirected to point to lists of subordinate level-two heads and, similarly, most level-two heads will be changed to point to lists of subordinate level-three heads, continuing in such a way until a level of head is reached that has no subordinate head. (The processing of the drill-down files is more sophisticated than I have let on. For example, a means was devised to preserve a way to navigate to those higher-level discussions that happen to contain one or more subordinate levels of heads.)

The lowest-level heads are those without any subordinate heads. They are left defined as FrameMaker created

them so that they take the reader into a particular book passage. Such book passages must correspond to WinHelp topics. To treat them as discrete WinHelp topics, another custom awk program strategically inserts the necessary markups (manual page breaks) throughout each chapter file for the book. As will be mentioned later, this MIF markup processor also implements some cosmetically useful font-size changes in the chapter files.

I let the MIF processor loose on the chapter files directly after FrameMaker has resolved cross-references and other computed book elements that they typically contain. Under UNIX, this can be accomplished by issuing an Update command by way of FrameMaker's `fmbatch` utility. Several shell command scripts orchestrate all processing in part by relying on `fmbatch`, which is available for UNIX versions of FrameMaker only.

So far I have described how the source text is organized in terms of a set of body files, one per chapter as is customary; and a set of drill-down files, one per header level.

The final compilation step has numerous options that enable and disable various Help browser features, such as whether graphics files will be compressed or not. In this regard, HyperHelp closely resembles WinHelp. Although professional programmers will be quick to assimilate the protocol for establishing compiler options, the casual user may be in for quite a struggle.

I was able to relieve my client from having to specify certain compiler options—such as the names of the MIF markup files—through some very basic UNIX file-manipulation commands. For example, I redirect a listing of MIF files generated by the `ls` command into a file whose name is mentioned in an *include* compiler directive that I supply in a boilerplate *project* file.

This dynamic way of specifying file options to the compiler eliminates human intervention and avoids filename synchronization errors. Why not let the shell scripts do the work based on the names of files you established when you set them up in FrameMaker?

Three book-building shell scripts orchestrate a very thoughtful sequence of processing, in the spirit of UNIX *make* scripts. Generally, there is one book-building shell script for each type of processing: (1) a book-building script for the chapter files that optimizes them for on-line viewing, (2) a book-building script for the generation of the drill-down files, and (3) a book-building script to invoke the compiler properly while affording a simpler interface to many compiler options.

A Little Relief from Compiler-Induced Madness

As you might expect, the HyperHelp compiler is as fussy as the WinHelp compiler. Each of these compilers requires syntactically correct markups in the textual source files. Each one chokes on the slightest typographical error in a control (project) file and related *include* files. Error messages are often cryptic or just plain misleading. In general, most authoring tools presume that you are familiar with the vagaries of software development environments, such as compiler directives.

The good news about the HyperHelp browser is that it has some helpful tracing functions that the Windows Help browser lacks. For example, it can show the context string associated with any recent hypertextual action that you performed. This facility helped me get past several seemingly insurmountable roadblocks.

Supplementary help-authoring tools typically hide the command-line and compiler-directive interfaces of the WinHelp compiler. Likewise, they often sit inside an extensible publishing tool such as Microsoft Word where they can simplify many markup-generation tasks. However, such tools make life easier only when they work without a hitch. The vendors of supplemental tools ought to offer technical support second to none.

Besides the trace facility, HyperHelp has one more virtue that I consider invaluable: HyperHelp can segregate the graphics inside MIF files into temporary external files just long enough for the compiler to finish its business with them. While graphics can be interspersed along with the text in RTF or MIF files, none of the compilers I have mentioned can tolerate them being there. This ordinarily leaves you with a whole bunch of file management chores. However, a HyperHelp user need not fuss at all. Well, maybe a little:

You may find that the UNIX `/tmp` directory seizes up during compilation due to an overload of multi-megabyte graphics files. In such an event, HyperHelp's error messages are very misleading. In the client-server environment I was using, I typically never saw the system-level "file system full" error messages at all. In any case, experience taught me to be suspicious of this particular problem after repeated compilation failures. (HyperHelp would be a much better UNIX citizen if it caught interrupt signals and cleaned up after any stray graphics files it had been creating in the `/tmp` directory.)

(For those of you keeping track of file counts, add 100 or so temporary graphics files to the 20 stock temporary files created by my custom shell scripts. Add 30 to the file count to roughly take into account all the permanent files, including shell scripts, related control files, and several HyperHelp startup and executable files. Organizing all these temporary and permanent files required a couple of directories per book, along with about 12 stock directories for the permanent files.)

Avoiding the Tool-Integration Blues

One way to improve the levels of integration between tools is to choose them so they conform to non-proprietary standards. Tools designed to work well with other tools typically can import and export data in standard formats, such as RTF. One FrameMaker virtue is its ability to export in an ASCII markup called MIF. MIF can easily be parsed with any UNIX text processing tools, including my favorite, the awk text processor. Admirably, the HyperHelp compiler accepts not only MIF files, it also accepts RTF files and SGML files.

Because my client's CAD tools worked on IBM, Sun, and other engineering workstations, I enjoyed abundant access to UNIX tools. With X-Windows to help UNIX serve

the FrameMaker and HyperHelp applications, I could move between systems readily without incurring a productivity hit. At a new workstation, I merely chose from a startup dialog the UNIX system where my application services would originate.

By taking full advantage of Frame's `fmbatch` utility for UNIX, my book-building scripts could take control all of the processing needed to optimize and compile on-line books, in cradle-to-grave fashion. Accordingly, these processes and tools are a real breakthrough.

MIF files do not have to be present initially for my tools to process them. The `fmbatch` utility permits them to be dynamically extracted by a shell script. The final processing step is the customary submittal of MIF (or RTF) files to the compiler, after which the MIF files are deleted. This shows how the on-line and hardcopy versions of the book are truly created from a single set of master files.

Even without my awk tools, the HyperHelp literature holds out the promise of superb automation when used in conjunction with FrameMaker. By itself, however, it falls shorts in several respects: (1) It expects you to manually save the MIF files to be compiled—it has no provision to call `fmbatch` as a way to request that FrameMaker generate these files automatically; (2) it expects that you won't mind if the entire book is converted into one oversize, long-scrolling WinHelp topic—or one that is broken into topics based on page breaks or manually inserted markups; (3) it makes no attempt to minimize the efforts to compile a book, nor to reduce the maintenance of the many required specifications for various compiler options; and (4) it does not offer drill-down navigation where you navigate through heads one level at a time.

The first three drawbacks have an deleterious impact upon the intensity of labor required to obtain professional-quality, on-line books. The final drawback deprives readers of a drill-down interface better suited to lengthy on-line reference works. (Windows 95 diminishes the amount of labor required to obtain drill-down styles of navigation, but even with the use of its navigational features, its labor requirements would have been prohibitive.)

Serving Two Media Masters

Once you go beyond the drill-down layers of navigation, the substance of the hardcopy and on-line books is identical—with only a couple of minor exceptions. One exception involved font-size changes for all levels of heads. While they would normally be formatted at different font sizes, my tools changed the MIF markup to impart to them the same appearance in the on-line book. Additional MIF markups were inserted to cause HyperHelp to freeze the opening headline for a topic in a window pane of its own. This ensures that readers will always have a way to identify the topic they are currently viewing.

The use of a consistent size head in a dedicated window pane lends the on-line book a more solid feel, because that window pane will always remain a fixed size. If you permit the topic headline's font size to vary for topics that are at different header levels, the reader will notice a jiggling of the window panes as they navigate between topics. For me, this goes over about as well as fingernails screeching across a blackboard!

Note also that the wide of range of font sizes suitable for a hardcopy book may detract from an on-line book. For one thing, the body font size needs to be boosted to promote its on-screen readability. However, to allow all the other font sizes to increase in roughly the same proportion has a tendency to create gargantuan font sizes for the highest levels of heads. Therefore, it is wise to shrink larger font sizes moderately, despite an increase in the body font size.

Technical Success, Marketing Failure?

After about 140 hours on this project, my client took delivery of a very professional-looking subset of on-line documents. A couple of weeks later, the client had leisurely converted another six or so titles. By then, an entire suite of books for the Tecnomatix tools that run on a single CAD platform had been converted, yielding 35 megabytes of browser files.

Each year, I suspect that thousands of WinHelp on-line documents will be created. By comparison, those created yearly for UNIX environments is minuscule. So even though I now have some of the best tools around, they are targeted at a relatively small market.

This helps explain why I am taking time out now to rework these tools to handle RTF forms of hypertext markup from programs other than FrameMaker, and to work with compilers from Microsoft in place of the HyperHelp compiler from Bristol Technologies.

Error messages are often cryptic or misleading. Most authoring tools presume you are familiar with the vagaries of software development environments...

Farewell for Now

As I step up to my latest challenge, who will step up to take the reins of "Fast Forthward"? As for me, writing this column has been a long and enjoyable journey into some of the bright and some of the dark corners of Forth. I am especially thankful for the support and encouragement I received, particularly from Marlin Ouverson.

Meanwhile, may the marketing forces be with each of you and may each of you have a happy and prosperous year.

—Mike

Fast FORTHward

Success Stories Sought

Mike Elola
San Jose, California

My enjoyment of technical writing peaks when I can contribute to the success of a publishing effort by writing programs. I was my happiest and my busiest when I was developing a publishing system tailored to the UNIX reference manuals and their associated on-line documentation systems. Those programs were shell and text processing scripts that provided rarely dreamed of levels of publishing automation.

Recently, my curiosity has taken me into the depths of AppleScript programming for Macintosh and UNIX-style programming under DOS. The latter is made possible with products from Thompson Automation Software (and other vendors). I have found their Tawk programming language to be an interesting variant of normal awk because of some of the unique extensions it offers. I intend to use it to process a markup known as Rich Text Format (RTF), and possibly HTML as well.

For someone with my background, one of the best areas of opportunity today is creating WinHelp and other on-line viewable documentation. On-line publishing with WinHelp resembles programming because it requires a markup language, RTF, and a compiler that converts that language into a corresponding binary file that the Help browser displays as on-line help or on-line books.

This has been a long, enjoyable journey into the bright and the dark corners of Forth...

This activity has already become the largest part of my consulting work. To focus my efforts towards developing on-line documentation, I will relinquish "Fast Forthward" with this installment.

So that I do not miss out completely on the discussion of Forth versus Java and other emerging topics, I will continue contemplating Forth in light of these developments.

I am going to leave you gentle readers with one of my own success stories about on-line publishing. It is offered to inspire you to follow my lead.

By showing how Forth has contributed to your success, you will play a part in the marketing of the Forth programming language. Trust me, readers will want to hear about it!

Flashback

During the walk from parking lot to office building, I realize that this day is another one of those perfect, summer-like, mid-September days in Silicon Valley. As much as I wanted to slow down to take in the sights and sounds, I had an equally strong urge to stare at several 21-inch computer screens in the classroom/lab where my client had settled me down to work.

My client wishes to publish a complete suite of reference guides for their product line, which consists of sophisticated CAD-application extensions. For example, several extensions work in concert to permit a robot arm to take measurements that can qualify or reject a custom-manufactured part in terms of its meeting previously captured tolerance specifications.

This project was bringing back the old excitement I once felt years earlier, but had lost. With the agile UNIX toolset to back my efforts, I felt confident about being able to solve any problems that lie in the way of converting FrameMaker-based hardcopy books to on-line-viewable documentation.

What a powerful incentive this effort held for me: If successful, I would be able to claim that I had automated the conversion of about five thousand pages of reference books for electronic distribution. However, I was duty-bound to explore all the possible solutions that did not involve custom programming. After all, the client wanted to limit the contract hours to no more than a few weeks!

Dutifully, I had spent the first half of that first week chasing down every obscure feature and menu option that might help make this conversion possible without the creation of any new tools.

With the dust from my research more settled at that point, I saw that even though the HyperHelp tool could take advantage of certain automation features found in FrameMaker, it was not enough. To reliably and efficiently impart high production values to these on-line books, more automation was sorely needed.

I am certain that my client did not realize the broad scope of the tools that might be built. Part of the role of the consultant is to try arrive at solutions that are more complete than the client imagined. The real allure for me was to create tools that address the on-line book conversion process from start to finish.

Imagine my predicament. I did not wish to throw too damp a towel over my client's expectations, yet I nevertheless had to confront him with the difficulty of our situation. I softened the blow considerably by offering to accept less than my full rate for hours of work I might need to provide beyond those in our original contract. I wanted to be certain that I would be involved in the project as long as really required. Also, by contributing some of my labor I could lighten my daily workload, avoid burnout, and enjoy the work more. With a less-than-breakneck pace, I tend to work more efficiently and more reliably anyway. (Will management ever get it?)

The first week came and went and I had no working prototype. I had to take a few days off for personal reasons, so I still had a for-pay week to go. That week came and went and still no prototype. Finally, at the end of a third week I had a working prototype. For my trouble, I would enjoy unhindered use of the newly developed code in future projects.

The prototype became fodder for a presentation that my supervisor used to sell upper management on a bigger, more legitimate project. Word finally came to me about a month after I completed the prototype that the project would resume.

For the duration of the second contract, my client and I were understanding each other better and functioning smoothly as a team. I no longer worried that my hard work might not be appreciated. Although we cleared some difficult hurdles during that time, this feeling that we could depend upon one another for support made all the difference.

Book Covers Revisited as Contents Topics

There are many unique aspects of on-line books that distinguish them from their hardcopy counterparts. In Win-Help parlance, the leading element of an on-line book (or help system) is a Contents topic.

The Contents topic displays the graphics and text that the reader will encounter first. It is typically brief because it functions as the cover of an on-line book. However, it also functions as a capsule contents section. That way, an initial set of hypertext links can help lead the reader to the passage they truly sought. Typically, the Contents topic lists the chapters available.

The visual appeal of a Contents topic is improved with a company logo or some other easily assimilated color graphic. However, the graphic should not eclipse the hypertext links.

Note that if the on-line document is really an on-line help system, it will have close ties with a software application. Because of this, a request for context-sensitive help initiated from the application will normally bypass the Contents topic, favoring a topic that is more relevant to the current operating context. In such a case, the Contents topic should probably be as task-oriented as possible.

When Automation Becomes a Necessity

Despite their daunting size and scope, the mainstream publishing programs sometimes come up short. For example, we could not rely on FrameMaker alone to generate a set of drill-down topics to augment the Contents topic. Still, this type of automation was sorely needed for my client's lengthy suite of reference manuals.

HyperHelp was the name of the browser and companion compiling system that we were using to build and view on-line books. It is one way to gain access to a WinHelp-style browser when running UNIX on engineering workstations like those that my client's CAD tools required.

While HyperHelp honors the hypertext links that FrameMaker builds when creating a table of contents, a Contents topic is not well served by that feature alone. For about five thousand pages of reference materials, a Contents topic built this way would occupy the screen space equivalent to 30 or 40 hardcopy pages.

Can you imagine a single WinHelp-style Contents topic that ran that long? As inappropriate as that would be, that was the state of the art in terms of cooperation between FrameMaker and HyperHelp.

Incremental, drill-down navigation avoids a massive Contents topic—displacing it with a hierarchy of coordinating navigation topics. For would-be publishers of large sets of reference documents, the lack of such a feature is tantamount to failure. (Indexes and keyword searches can also be important forms of navigation.)

The bad news with respect to a drill-down style of navigation is the extensive labor involved in its creation. Custom tool development was therefore more than justified. Neither enough staff nor enough time could be mustered for the hand-assembled approach, as I am about to explain.

Suppose you just finished creating a few thousand links. Usually, you will not immediately know which fifty or so links are broken due to incorrect specification of a topic destination or of a hypertext link's markup. The compiler usually complains about the latter, but not the former type of error.

(Even when it does complain, the compiler reports about a topic number—as though you were maintaining sequentially numbered topics in your source files. Yeah, sure. Even if you use one of the third-party authoring tools, their amenities tend to taper off at the compiling and debugging stages. For compiler errors that refer to topic numbers, no mapping of these numbers to human-friendly names of topics will be made for you. You'll be lucky if the tool can map them for you at all.)

Assume you suspect there are link errors in the successfully compiled book. You have no choice but to hunt for them in the generated on-line book using the compiler's companion browser. Furthermore, you will have to interactively test many, if not all, of the good links

(Continues on page 39.)

The Institute for Applied Forth Research, Inc.
Announces the 16th Annual

1996 Rochester Forth Conference on Open Systems

June 19 – 22, 1996
Ryerson Polytechnic University
Toronto, Ontario, Canada

Call for Papers

The 1996 Rochester Forth Conference on Open Systems is hosted by the Institute for Applied Forth Research, Inc. in conjunction with the Southern Ontario Forth Interest Group and McMaster University. The Rochester Conference will again provide a forum for researchers, developers, and vendors to present the latest practical results dealing with open systems. The conference seeks original papers relevant to the design, development, implementation, and use of open systems. Conference topics include:

- Open Firmware Standard/Open Boot™
- Scripting Languages
- Distributed Computing
- Plug and Play™ Systems
- SGML and HTML
- Educational Issues
- Java™

Other areas of interest are Forth programming standards, embedded systems, real-time systems, and the use of Forth for scientific and engineering applications.

Important Dates

February 1, 1996	Deadline for an extended abstract
May 1, 1996	Deadline for the camera-ready copy of final paper

Please send all manuscripts to the Program Chairman.
Additional information will be posted on the world-wide web as it becomes available:
http://maccs.dcss.mcmaster.ca/~ns/96roch.html
Facilities Chair: B.J. Rodriguez

Program Chair
Nicholas Solntseff
Dept. of Computer Science & Systems
McMaster University
Hamilton, Ontario
Canada L8S 4K1
ns@maccs.dcss.mcmaster.ca

Conference Information
Lawrence P.G. Forsley, General Chair
Institute for Applied Forth Research, Inc.
Box 1261
Annandale, Virginia 22003
fax: 703-256-3873 phone: 716-235-0168
lforsley@jwk.com