VOLUME X, NUMBER 1 MAY/JUNE 1988

FORTH DIMENSIONS

LOCATING FORTH WORDS

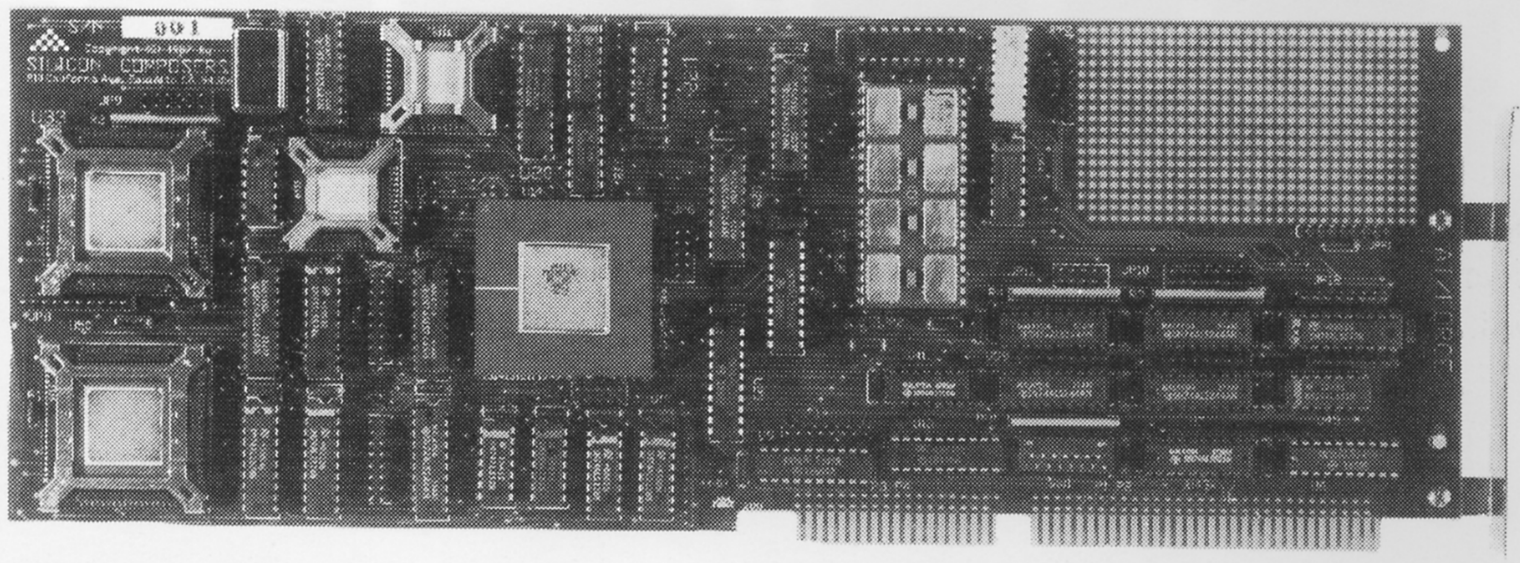
ACCESS TO dBASE FILES

IMPROVED STRING HANDLING

ANS FORTH UPDATE

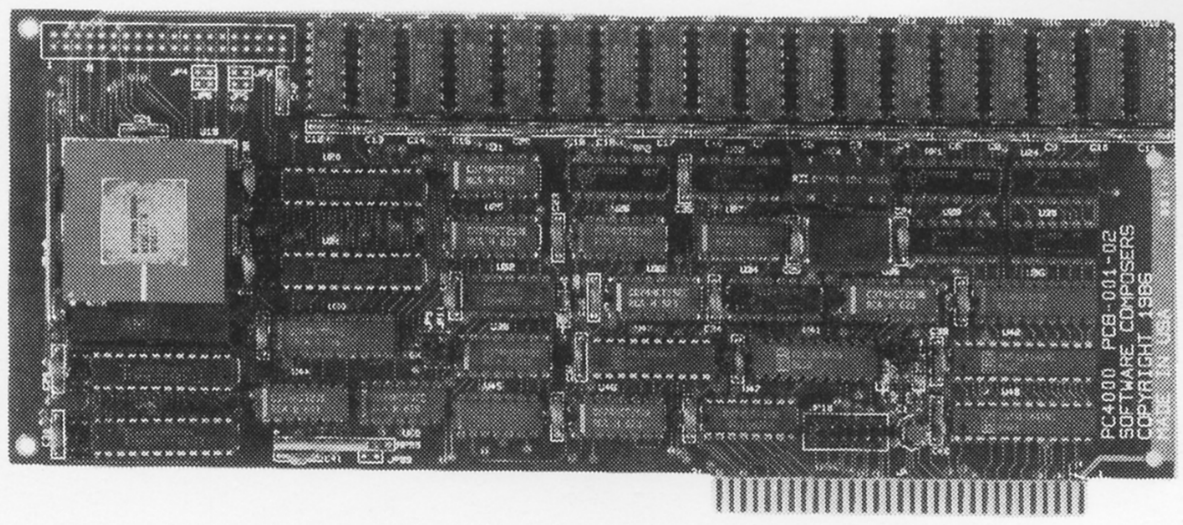
SUPERFAST FORTH SYSTEMS

AT/FORCE COPROCESSOR: 10 TO 50 FORTH MIPS



- Five chip Harris Core Set:
  10 MHz Forth RISC Core
  1-cycle 16 × 16 multiplier
  1-cycle 15-channel interrupt
  two 64-word stack controllers
- Plugs into AT
- Forth Software included
- 32K bytes main memory
- Expandable to 128K bytes
- AT shared memory space
- 2″ × 3″ prototyping area
- All Core signals available
- Runs concurrently with AT
- 2 weeks ARO: $3995

PC4000 COPROCESSOR: 4 TO 10 FORTH MIPS



- NC4016 Novix Forth Engine
- Plugs into PC/XT/AT/386
- 512K bytes main memory
- PC shared memory space
- Multiple PC4000 operation
- Forth Software included
- NC4016 signals available
- Runs concurrently with PC
- Compact ¾ length format
- 2 weeks ARO: $1295

Both ideal for real-time control, data acquisition and reduction, image or signal processing, or computationally intensive applications. For additional information, please contact us at:

SILICON COMPOSERS, 210 California Ave., Palo Alto, CA 94306 (415) 322-8763

SILICON COMPOSERS

FORTH DIMENSIONS

EDITORIAL

About those standards: I don't really understand how a new language standard arises out of "common usage." I suppose common, as it is used here, means both unremarkable and general, so we are all safe. But if so, that means everyone on the ANS Forth Technical Committee could be surprised in a big way if one or two of the larger dialect communities — who may even be more *common* than the rest of us — organized to show that their users mean business, too.

Changes to Forth will appear in the next standard through the agencies of the humans involved. Those who think an ANS stamp will help their business or career owe it to themselves to get one. Those who fear the consequences of the ANS standardization effort — which will, after all, go on without them — should get involved if only to ensure that their views are taken into account. Which is reason enough to print in this issue the pages needed to submit technical proposals to the ANS Forth team.

Name your favorite Forth reference works: I'd be among the first to rejoice if the last word in Forth standards could be uttered. How nice it would be to have one dialect, indivisible. It is, understandably, quite a challenge to keep track of the words allowed by the three fundamental dialects, plus the additional words in (or derived from) MVP, LMI, poly, MMS, F83, and others.

My position is that it's better to publish a reasonably broad range of material than to restrict our published content to an easily filtered, standard dialect. And I do not believe that code blindly typed in, even if it runs, contributes much to the expertise of the typist. The people I've met who succeed in Forth careers have been conversant with (or, at least, calm in the face of) a variety of dialects. Because, for better or worse, that's the way of the working world.

A reader laments in this issue about the undefined-word problem (an undefined, non-standard word, or assembler of the wrong flavor, cropping up in source screens). Our authors should all know that no one wants undefined words in his source code. Words that don't adhere to a standard dialect need to be defined. Routines containing assembler code should be documented in high-level Forth as well as in English. I think the solution is to continue doing our best, and to help readers educate themselves about the Forth implementations they encounter.

We can, however, do a better job of guiding people with questions to the best sources of answers. Send me a list of the reference works you use to look up Forth definitions, techniques, etc. I will organize the responses into a "Recommended Reference Library" for Forth programmers and see that it gets placed where it will do the most good.

A friendly reminder: This magazine is by and for its readers. We're interested in your latest project, the trick you used to complete an application, your experiences with Forth (especially on *today's* machines), programming techniques, and, yes, *FD* is looking for a few good routines.... In other words, just about every one of you has something to contribute to these pages. If you haven't noticed, the excellent *Bibliography of Forth References* shows that you would be in good company. If you're a new author, send me your name and address to get on the list to receive our soon-to-be-revised writers' guidelines. (But the bottom line is: start writing!)

—Marlin Ouverson
Editor

Forth Dimensions
Published by the
Forth Interest Group
Volume X, Number 1
May/June 1988
Editor
Marlin Ouverson
Advertising Manager
Kent Safford
Design and Production
Berglund Graphics
ISSN#0884-0822

Forth Dimensions welcomes editorial material, letters to the editor, and com-ments 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 $30 per year ($42 overseas air). For membership, change of address, and to submit items for publication, the address is: Forth Interest Group, P.O. Box 8231, San Jose, California 95155. Administrative offices and advertising sales: 408-277-0668.

About 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* is published bimonthly for $24/36 per year by the Forth Interest Group, 1330 S. Bascom Ave., Suite D, San Jose, CA 95128. Second-class postage pending at San Jose, CA 95101. POSTMASTER: Send address changes to *Forth Dimensions*, P.O. Box 8231, San Jose, CA 95155."

LETTERS

Search Party
Dear FIG:

I was pleased to see the response by Mr. Martin Guy (*FD* IX/4) to my letters and "search" code of several months earlier. When I submitted my code, I hoped just such a response would be generated.

I am glad my efforts have been helpful, and I appreciate the improvements he has suggested. I accept his critique regarding `BEGIN ... UNTIL` and `BEGIN ... WHILE ... REPEAT`, fully understanding why only after carefully thinking it through. *His code is visibly clearer and more to the point.* I especially like Mr. Guy's use of the anonymous variable to hold the normalized initial string character, thereby saving repeated testing and converting.

However, I have one re-improvement to re-place into his version.

One feature of my solution was that I did not have to reload the initial string character after every comparison for a match, I only reloaded (and gratuitously retested/converted) in the event that there had been a match on the first character, with a subsequent failure to match on the entire string.

To reimplement this feature, simply move the `4 ROLL #) AH MOV` instruction, which presently begins the `WHILE` portion of the outer loop, to the line between the two `THEN` directives, changing `4 ROLL` to `6 ROLL` in the process; and add `AH AL XCHG` immediately before the `BEGIN` of the outer loop. By doing so, we add only two bytes to the code size for the `XCHG` instruction, but we gain (according to my timing information) ten clocks each time there is no match on the first character of the string. Intuitively, I would expect that, even when searching for the letter 'e' (the most common letter in the English language), there would be more failures to match than there would be matches. For a buffer of 1K, this savings could easily approach 8000 - 9000 clocks or more during a search of the full buffer. Real savings.

Sincerely,
Robert Lee Hoffpauer
609 W. Arapaho Road
Richardson, Texas 75080

Fractals, Windows, and 64K
Dear Sir,

Further to my previous letter, it seems I always write for help too soon, and then ask all the wrong questions! Please accept my apologies for even suggesting there may be some typographical errors in the code published for "Fractal Landscapes" — I found out what +- is supposed to do from a copy of the fig-FORTH installation manual's glossary, and the result was just what Phil Koopman, Jr. intended. Terrific stuff!

The fact, however, that F83 limits code space to 64K meant that it was necessary to set the graphics array up in virtual memory on disk, and run the whole process from my RAMdisk.

This is OK, it works well — I set up a small array of colour codes to change the order in which the colours are accessed, which gives a more pleasing display, but it's still too slow. The question I should have asked is how to access all that beautiful memory from F83? Perhaps someone could steer me in the right direction here?

In addition, I have been experimenting with F83's multi-tasker with limited success. I note that Dr. Ting, in his *Inside F83*, suggests that "...windows are built this way." I wonder if an article in *Forth Dimensions* elaborating on this theme might be of interest to those of us who are still struggling with Forth? It would certainly be appreciated by me!

Yours faithfully,
David J. Dartnall
20 Eldon Street
Dianella 6062
Perth, Western Australia

Thanks for writing, David. Your shifting areas of confusion seem to be following a pattern familiar to many self-taught Forth programmers. One of the best places to look for the definitions of unfamiliar words is Glen B. Haydon's All About Forth. *Ah, windows... I agree with your assessment of their interest to most of us — off-the-cuff, I'd say the code and documentation you want is in Volume Two of* Dr. Dobb's Toolbook of Forth; *but I'd like to see an article that addresses the technique. As to those 64K blues: it gets tricky, and maybe it's best to talk to some of the folks at your local FIG chapter. (All right, I know Melbourne and Sydney aren't local to you, but I'll bet they can put you in touch with someone closer...) You could develop a memory manager that uses far calls instead of jumps, and then add 32-bit addressing; or you could work around it by keeping your program within the 64K limit, but putting Forth libraries and data out into different 64K segments. (The latter approach is used to good end in LIB-FORTH, a four-diskette set from Mountain View Press.) Good luck!*

Open Letter to Standards Committee

This letter is in response to a request I, as chairman of the Boston chapter of the Forth Interest Group, received from a

member of the ANS Forth standards committee [X3J14] for input into the standards process. This is not a new subject for our group. Over the past year-and-a-half, we have discussed it at length at least three times, as well as following on-going developments. Our chapter represents a cross-section of New England's Forth community. Meetings normally include several consultants, several Forth companies, vendors, and even a few hobbyists!

This request spurred me to bring the matter up at our next meeting. The opinions presented in this discussion basically fell into two camps. One felt we should not get involved, in hopes that nothing would ever come of it. The second group felt we should get involved, in the hope that we could prevent too much damage. All who participated in the discussion seemed to regard a new standard as a Pandora's Box that shouldn't be opened.

In addition, the discussion touched on what should or shouldn't be in the standard. In this small, friendly group, we could not reach *any* concensus as to what should be included, to say nothing about the details of how it should work (that's easy — it should work like *my* system).

At this time, there doesn't seem to be *any* grass-roots support for a standards effort. I regard this as a major crisis which the committee should address immediately. The Forth community needs to be convinced that their opinions will be listened to; that the resulting standard will not invalidate their systems; that arbitrary and unnecessary changes will not be made; and that the results will reflect "common usage" and will not try to "improve" the language.

If we are hoping to generate a standard which the Forth community will accept and appreciate, we must first bend over backwards to convince that community that we are working for it. It is not good enough to invite input, we must go out and seek it. To determine what is "common usage," we must talk to the "common users," not the vendors. We must not ask for input on non-existent forms, but instead circulate questionnaires to users. It is not good enough to open meetings to the public, key members of the committee should also visit local chapter meetings on a regular basis.

The alternative is to promulgate another divisive "standard" which only serves to fragment our community. The end result of this will be reflected in *more* resistance from companies when we suggest using Forth. The ANSI rubber stamp isn't worth the cost at this price!

Gary Chanson
360 Waltham Street
W. Newton, Massachusetts 02165

Code for Local Variables

Dear Editor,

The local variables in the article by Peter Ross (*FD* IX/4) suffer from a major defect: they cannot be called recursively. The use of recursive calls is one of the major reasons for having local variables. The letter by Henning Hansen (*FD* IX/5) certainly cures that and gives an altogether better approach to local variables. Mr. Hansen suggested that his procedure could be made into an assembly language procedure to speed it up. The enclosed code does just that; for LMI's PC/Forth, at least. I have renamed Mr. Hansen's `(LOCAL)` procedure to `RESTORE`, since that is what it does: restore the value of the variable. I have also modified it so that it can be used either with variables or with quantities (a data type that fetches its value like a constant when invoked, but which can have that value changed; this type is not favored by the Forth establishment, but it has certainly made my programming easier). This changes Mr. Hansen's syntax so that the `LOCAL` declaration precedes the name of the variable or quantity.

The user who prefers Mr. Hansen's syntax may simply omit `LOCAL`, rename `local` as `LOCAL`, and use it directly. The compiled code is almost the same, and the execution speed using a `VARIABLE` is indistinguishable, although still slower than using a `QUANTITY`, as seen in the timings below.

In Mr. Hansen's procedure, the speed penalty of using recursion with local variables was a factor of approximately 6:1 on my system (because on the 80296, the multiplication is vastly speeded up, which helps the faster, non-recursive procedure proportionately more). While using the assembly language version, the penalty is a factor of less than four, and even less using a `QUANTITY`. In fact, the timings for the three tests were 9.23 seconds for 10000 iterations of 12 N!, 8.40 with n!, and 2.42 for M! on the accompanying screens.

Sincerely,
Michael Barr
146 Dobie Avenue
Mount Royal, Quebec
Canada H3P 1S4

And NEXT,
in No Time at All

Dear Marlin,

I am glad that Carl A. Wenrich profited from the tutorial on the `NEXT` function in *All About Forth*. Lest he be satisfied with his faster `NEXT` loop (*FD* IX/6), he should know that our latest `NEXT` in the WISC CPU/32 takes zero time. This is, perhaps, the ultimate in speed.

Sincerely yours,
Glen B. Haydon, M.D.
La Honda, California 94020

```
Screen # 117
\ local variables                                        12:50 03/05/88
ASM
CODE restore    BX, [BP] MOV      AX, 2 [BP] MOV
                [BX], AX MOV      BP, # 4 ADD
                NEXT, END-CODE
: RESTORE      restore ;
CODE local      BX POP CX, [BX] MOV -2 [BP], CX MOV
   -4 [BP], BX MOV  WORD   -6 [BP], # ' RESTORE >BODY MOV
               BP, # 6 SUB    NEXT, END-CODE
: LOCAL ' >BODY [COMPILE] LITERAL COMPILE local ; IMMEDIATE
-->

Screen # 118
\ test local variables                                   12:59 03/12/88
13 LOAD                 \ timer module
VARIABLE VAR QUANTITY QUAN
: N!     LOCAL VAR
         DUP VAR ! 0= IF 1 ELSE VAR @ 1- RECURSE VAR @ * THEN ;
: n!     LOCAL QUAN
         DUP >> QUAN 0= IF 1 ELSE QUAN 1- RECURSE QUAN * THEN ;
: T !TIMER 10000 0 DO 12 N! DROP LOOP .TIMER ;
: t !TIMER 10000 0 DO 12 n! DROP LOOP .TIMER ;
: M! DUP 1 < IF DROP 1 ELSE 1+ 1 SWAP OVER DO I *  LOOP THEN ;
: T1 !TIMER 10000 0 DO 12 M! DROP LOOP .TIMER ;
```

TAKE-OUT FORTH

THE DSC-12:

2 Mhz R65F12 CPU
Laser trim D/A
Fast A/D with built-in track and hold
Rechargeable battery back-up
RS-232, RS-422, RS-485
Super real time clock
Signals amplifiers
40 I/O with timer and interrupt logic
High density board 7.5" X 5"
Expansion edge connector
On board Eprom, EErom programming

Full screen editor, interactive window
Assembler / Deassembler
Resident FORTH including words:
A/D@, CLOCK@, ?KEY, etc...
Full documentation (300 pages)
Program library on disk

$499.00

Special introductory offer includes:
Intelligent LCD display
20 keys Keyboard with encoder

Dianax INC.

VISA, MC, OEM welcome 514-878-2802
6899 Irwin, Suite 5, Montréal, Qc, Canada H4E 4L2

TI-FORTH, Forth-79

LOCATING FORTH WORDS

GENE THOMAS - LITTLE ROCK, ARKANSAS

I've stopped hunting through disks, screen by screen, for a word I haven't looked for in a long time. Now I let the machine do the work. One of the strong points of a computer is that it will perform boring and repetitious tasks all day long without complaint.

Give SHOWME a range of screens followed by a string, and it will search those screens for that string. The designated string can be a single word or a phrase. Because you set the string delimiter with a <cr>, it can even be a substring.

Let's assume this code is on a screen in the specified search range:

```
: SATISFY
  KOOL-AID @ COLD MAKE
  DOWN GULP ; IMMEDIATE
```

All of the following will work:

```
lo hi SHOWME COLD <cr>
lo hi SHOWME DOWN GULP <cr>
lo hi SHOWME KOOL<cr>
```

Of course, searching for a string rather than a discrete word has its down side. SHOWME may, at times, find a matching string that is not what you intended. For example, suppose you designated the string COLD for the search. And further, suppose that on the screen preceding the one on which COLD appears is the word COLDER. That's a minor problem which I think is subordinated by the convenience of being able to designate any string. "False finds" can be avoided by adding spaces to the beginning and end of a string that is a discrete word, because the spaces become a required part of the match.

Listing one contains the Forth code for the SHOWME utility. I believe that most of the code is pretty straightforward, so in-depth comments will be limited to SEARCH and MATCH?. The code was developed in TI-FORTH, but it is conventional fig-FORTH and, therefore, very portable. For those who are operating with Forth-79 or Forth-83, here are some tips on conversion. (Actually, very few changes will be necessary.)

1. If your WORD leaves an address on the stack, then add DROP as the last word in the definition of STRING.
2. Nearly all Forths have the word WHERE. If yours doesn't, then define DISP-SCRN as `: DISP-SCRN DROP PAGE DUP ." Screen #" . LIST QUIT ;`
3. If you don't have the user variable CURPOS, or another variable which holds the cursor position to substitute, then define the word POSIT (position) as `: POSIT CR ." Searching screen #" . ;`

The only two definitions of any real complexity are MATCH? and SEARCH. SEARCH loops through the specified

SHOWME Locating Utility

Listing 1

```
Beginning scr #83
  0. \ SCR #1: SHOWME                                  Gene Thomas, Oct85
  1. : DISP-SCRN  \ scr# IN -- ;display screen containing string
  2.     2+ SWAP  DUP   PAGE  ." Scr #"  .    WHERE   ;
  3. : NOT-FOUND  \ --
  4.     CR  STRADR TYPE   ." Not found."    QUIT  ;
  5. : POSIT  \ scr# -- ;something to look at while waiting
  6.     CURPOS @ SWAP  ." Searching scr # "  .    CURPOS !  ;
  7.
  8. : MATCH?  \ adr1 adr2 cnt -- f ;true if strings at adr's match
  9.     0 SWAP 0 DO  DROP      \ drop dummy flag 1st time thru (zero)
 10.                OVER I + C@   OVER I + C@   -
 11.                 IF  FALSE LEAVE                \ non-matching char
 12.                  ELSE  TRUE                    \ char matches
 13.                 THEN
 14.              LOOP   >R DROP DROP R>  ;         \ drop adr1 & adr2
 15.   -->
 16. \ SCR #2: SHOWME                                        GT, Oct85
 17. : ASCII?  \ block-adr -- loop-index ;char#1 on line#0 printable?
 18.     DUP C@ DUP  31 > SWAP 127 < AND  IF B/BUF ELSE FALSE THEN ;
 19. : SEARCH  \ scr-hi+1 scr-lo -- ;search thru screens for string
 20.     CR
 21.     DO I BLOCK   DUP I POSIT                   \ loop thru screens
 22.       ASCII? 0 DO  DUP                         \ loop thru block
 23.                  I +  STRADR  MATCH?
 24.                   IF ( string found) DROP  J I DISP-SCRN  THEN
 25.                LOOP  DROP ( block adr)
 26.     LOOP ( string)   NOT-FOUND  ;
 27. : SHOWME  \ scr-lo scr-hi -- scr-hi+1 scr-lo ;takes following
 28.     STRING                                     \ string
 29.     1+ SWAP SEARCH ;
 30.
 31. ;S END                                                  SHOWME
```

screens via the word BLOCK which leaves the address of the disk buffer where the screen resides. In SEARCH's inner loop, the screen address (I +) and the address and character count of the string to be matched (STRADR) are fed to MATCH?. Any string whose first character is a non-printable ASCII value will be skipped. The buffer (screen) is looped through in the inner loop, unless the first word ASCII? feeds to the inner loop either 1024 (B/BUF) or zero. If the string match is found, then the screen number and offset into the screen are fed to DISP-SCRN, otherwise the outer loop brings up the next BLOCK.

As noted above, MATCH? receives the address within the BLOCK to begin searching, the starting address of the search string, and its character count. MATCH? then loops through the count, comparing the corresponding ASCII numbers in each byte. By subtracting one ASCII value from the other, it determines a match by the resulting zero. Because IF executes on non-zero numbers, a true flag will be left by ELSE as long as character matches occur. At the first non-matching instance, if there is one, a non-zero flag is left for IF and the loop is exited via LEAVE. (Why don't all of these flags pile up on the stack as matching characters are found? The answer is the explanation for the "dummy" 0 flag preceding SWAP. Each iteration of the loop performs the DROP following DO. The dummy flag provides something to drop the first time through.) After the looping is finished, via LEAVE or by exhaustion, the two remaining addresses are dropped, leaving only the resulting true or false flag.

Returning now to SEARCH, the flag left by MATCH? is acted on by the IF in the inner loop. A true flag invokes DISP-SCRN. A false flag causes the loop to iterate. If the inner loop exhausts itself with all repetitions of the outer loop, then the "not found" message will be displayed.

The three non-standard words in the listing are 0 CONSTANT FALSE, -1 CONSTANT TRUE, and PAGE. PAGE is my word for clearing the screen and homing the cursor.

Gene Thomas edits the Comment Line, *the newsletter of the Central Arkansas FIG Chapter, and is a registered polysomnographic technologist at the Sleep Disorders Center at the University of Arkansas for Medical Sciences.*

Forth-83

ACCESS TO dBASE FILES

JOHN E. TARIN - ROSSLYN, VIRGINIA

This article presents a generalized method to access data stored in dBASE files. (For those not familiar with the file formats of the popular dBASE series of data base and file management programs, see *File Formats for Popular Software* by Jeff Walden, Wiley Press, 1986; and the dBASE manual.) The mechanism retrieves information about any dBASE file from the header bytes of the file, and uses this information to locate data in the fields of each record. The words constituting the mechanism may be used to fetch data from more than one dBASE file during a session.

I wrote this program-fragment because I had to perform an analysis of the data stored in several dBASE files. Although dBASE provides an access-and-manipulation language, Forth is more straightforward, is easier to write and debug, and the completed code executes much faster.

The code presented here is written in LMI's PC/Forth+ (a 32-bit implementation). This Forth version was used only because the data files in question contain financial data, and the calculated values exceed the range of 16-bit integers. This choice eliminated the need to generate a numeric manager.

The procedure to read a dBASE file involves setting the filename, CREATEing a sufficiently large buffer (the number of bytes in the dBASE file can be used as a guide), and using the word PREPARE to read the file header information and data into the buffer. The word PREPARE takes as input a file's "handle_parms" and the target buffer address.

If the dBASE file is too large to be read completely, or if the user does not wish to read all records at once, the words READY

Table One:
File Access Words

READY
Reads the header bytes plus a non-data byte preceding the first record.

READ.DATA
Reads* all the data from the specified file into the buffer specified with READY.

READ.REC
Reads* the contents of one record at each read. This word is useful should the file be too large to read at once, or should the user wish to read records singly.

**Note:* A modified read function may be needed to bypass the DOS limit of 32K. In Forth versions with the updated READ function, a modified function is not needed.

PREPARE
This word combines the functions of READY and READ.DATA to provide simple input of an entire file, including header information.

SKIP.HDR
Used after a file rewind (i.e., 0 SEEK-ABS) to position the next read at the first data record.

SKIP.RECS
Used to locate the nth file record. (This word uses SEEK-REL.)

File Description Words

REC.CT
Number of records in the dBASE file. (Range is zero to REC.CT - 1.)

HDR.LEN
Number of bytes in each record.

REC.LEN
Number of bytes in the dBASE header.

#FLDS
Number (calculated) of data fields in each record. (Range is 1 to #FLDS.)

and READ.REC may be used to read the header information and any individual record. (In this case, the minimum buffer size will depend on the record size and the size of the header.) The word READY takes as input the file's "handle_parms" and the target buffer address, and reads the file's header bytes into the specified buffer.

The word READ.REC takes the file's "handle_parms" and an index, i, as input and reads the next available record into the ith record space of the same buffer specified for READY. With READ.REC alone, the user can read each record individually. If the user wishes to read a specific record, the word SKIP.RECS can be used to skip to the nth record in the file.

When a file's header information and records have been read, file-descriptive data in the header can be retrieved using one or more of a set of utility words. For example, record count, record length, field names, and field types can be retrieved using the words REC.CT, REC.LEN, FIELD.NAME, and FIELD.TYPE, respectively. In addition, calculated values such as the number of fields and field offsets within a record can be retrieved using the words #FLDS and FIELD.OFFSET. Once data from a file have been read and/or manipulated, the words in the package can be reused to read and access data from any other dBASE file.

Once data have been read, the user can switch attention between the data sets by resetting the address stored in HDR.ADDR. The word HDR.ADDR performs a function similar to an execution vector. You set it to the address of a buffer containing a dBASE header and data, and thereafter all access words will function for the file described by that buffer.

Table One is a partial glossary of access words. Table Two is a listing of the dBASE access fragment. Table Three is a listing of code using the fragment words to read and combine data from two dBASE files. Tables Four-a and Four-b show the structures of the two demonstration files, using the dBASE command "LIST STRUCTURE."

(Tables Two and Three follow.)

Buffer Definition Words

HDR.ADDR
Variable used to store the address of the currently used input buffer. May be changed at will to provide access to the contents of any buffer filled using READY and READ.DATA.

DATA.ADDR
Returns the address of the first data byte in the currently used input buffer.

Field Description Words

FIELD.NAME
Returns the address of the string containing the nth field name (10 bytes).

FIELD.LEN
Returns the length of the nth field.

FIELD.OFFSET
Returns the word address of the calculated offset for the nth field.

dREC
Returns the address of the beginning of the nth data record.

dFIELD
Returns the address of a specified field within a specified record.

LAST.RECORD
Returns the address of the last data record.

.STATS
Used to returns the name and length of the nth field.

Table Four-a:

Structure for database: C:wpdrep2s.dbf
Number of data records: 308
Date of last update: 05/04/87

Field	Fieldname	Type	Width	Dec
1	WPD	character	7	
2	FY86	numeric	8	3
3	FY87	numeric	8	3
Total:			**24**	

Table Four-b:

Structure for database: C:wpdrep5s.dbf
Number of data records: 258
Date of last update: 02/02/87

Field	Fieldname	Type	Width	Dec
1	WPD	character	4	
2	FY88	numeric	8	3
3	FY89	numeric	8	3
4	FY90	numeric	8	3
5	FY91	numeric	8	3
6	FY92	numeric	8	3
Total:			**45**	

Screen # 15

```
( dBASE HEADER ACCESS  -  INPUT BUFFER          14:41 04/30/87 )

: WALL ;

VARIABLE  HDR.ADDR
\ Holds the address of the data buffer currently in use.

\ dBASE HEADER ACCESS   Bytes  0 - 31

: VERSION  ( -- addr)  HDR.ADDR @ ;
: DATE  ( -- addr)  HDR.ADDR @  1+ ;
: REC.CT  ( -- n)  HDR.ADDR @  1+ 3 +  W@ ;  \  REC.CT < 10000
: HDR.LEN  ( -- n)  HDR.ADDR @  1+ 3 + 4+  W@ ;
: REC.LEN  ( -- n)  HDR.ADDR @  1+ 3 + 4+ 2+  W@ ;
: RESERVED  ( -- addr)  HDR.ADDR @  1+ 3 + 4+ 2+ 2+ ;
```

Screen # 16

```
( dBASE HEADER ACCESS   Bytes  0 - 31           14:41 04/30/87 )

: FIELD.DESC.OFST  ( -- n)  1  3 + 4+ 2+ 2+ 20 + ;

: #FLDS  ( -- n)  HDR.LEN  FIELD.DESC.OFST  /  1- ;

: HDR.LENGTH  ( -- n)  HDR.LEN  1+ ;

: DATA.ADDR  ( -- addr)  HDR.ADDR @  HDR.LENGTH + ;
```

Screen # 17

```
( dBASE HEADER ACCESS   Bytes 32 -  n           14:41 04/30/87 )

  0  DUP  CONSTANT  FLD.NAME.OFST
 11  DUP  CONSTANT  NAME.LEN  +
     DUP  CONSTANT  FLD.TYPE.OFST
  1  DUP  CONSTANT  TYPE.LEN  +
     DUP  CONSTANT  FLD.DATA.ADDR.OFST
  4  DUP  CONSTANT  FAD.LEN  +
     DUP  CONSTANT  FLD.LEN.OFST
  1  DUP  CONSTANT  FL.LEN  +
     DUP  CONSTANT  FLD.DEC.CT.OFST
  1  DUP  CONSTANT  FDC.LEN  +
     DUP  CONSTANT  RES.BYTES.OFST
 14  DUP  CONSTANT  RES.BYTES.LEN  +  DROP
```

Screen # 18

```
( dBASE ITEM ACCESS                             14:41 04/30/87 )

: FIELD  ( n -- addr)  FIELD.DESC.OFST *  HDR.ADDR @  + ;

: FIELD.NAME  ( n -- str.addr)  FIELD  FLD.NAME.OFST + ;

: FIELD.TYPE  ( n -- c)  FIELD  FLD.TYPE.OFST + C@ ;

: FIELD.LEN  ( n -- c)  FIELD  FLD.LEN.OFST +  C@ ;

: FIELD.DEC.CT  ( n -- c)  FIELD  FLD.DEC.CT.OFST + C@ ;

: FIELD.OFFSET  ( n -- w.addr)  FIELD  RES.BYTES.OFST + ;
```

Screen # 19

```
( dBASE ITEM ACCESS UTILITY                     14:41 04/30/87 )

: .FNAME  ( n -- )  FIELD.NAME  NAME.LEN  TYPE ;

: .FLEN  ( n -- )  FIELD.LEN  . ;

: .STATS  ( n -- )  DUP  .FNAME  .FLEN ;

: LAST.RECORD  ( -- addr)
       REC.CT 1-  REC.LEN  *  DATA.ADDR  + ;

: dREC  ( rec# -- addr)  REC.LEN  *  DATA.ADDR + ;

: dFIELD  ( rec# field# -- addr)
       SWAP  dREC  SWAP  FIELD.OFFSET W@  + ;
```

Screen # 20

```
( READ dBASE FILE HEADER                        14:41 04/30/87 )

: SAVE.OFFSETS ( -- )  0  #FLDS 1+  1  ?DO  DUP
       I FIELD.OFFSET W!  I FIELD.LEN  +  LOOP  DROP ;

: SKIP.HDR  ( handle.parms -- )  HDR.LENGTH  SEEK-ABS  DROP ;

: READ.HDR  ( handle.parms -- n,f/f)  32  HDR.ADDR @  READ ;
\ Reads the dBASE file-defining bytes 0 - 32

: READ.HEADER  ( handle.parms -- n,f/f)
       HDR.LENGTH  HDR.ADDR @  READ ;
\ Reads the complete header information from byte 0
```

Screen # 22

```
( READ dBASE FILE HEADER                        14:41 04/30/87 )

: READ.DATA  ( handle.parms -- n,f/f)
     REC.CT  REC.LEN  *  DATA.ADDR  READ  0=  ?DROP ;

\ NOTE:  READ.DATA reads the entire file.

: READ.REC  ( handle.parms i -- n,f/f)
     REC.LEN *  DATA.ADDR  READ ;

\ NOTE:  READ.REC reads one record into the i-th record space
\        of the user's buffer.

: SKIP.RECS  ( handle_parms n -- )  REC.LEN *  SEEK-REL  DROP ;
\ NOTE:  First  REWIND (0 SEEK-ABS)  &  SKIP.HDR.
```

```
Screen # 23
( READ dBASE FILE                                14:41 04/30/87 )

\ Code must be generated to name & open a dBASE file,
\ and to CREATE a sufficiently large buffer.

: PREPARE  ( handle.parms buffer -- )
        >R  2DUP  R>  READY  2DUP  READ.DATA  CLOSE.FILE ;

 \ PREPARE reads dBASE data into a user specified buffer
 \ which can be accessed using words on screens 15 thru 19.
 \ The data is input using words on screens 20 thru 22.
```

```
Screen # 25
( BUFFER FOR dBASE FILE HEADERS                  14:41 04/30/87 )

: WALL ;
\ The following code must be generated for each dBASE file
\ of interest.

10000  CONSTANT  dHDR1.LEN
\ Any buffer must be large enough to hold the entire dBASE
\ file with header (used with READ.DATA), or at least one
\ record with header (used with READ.REC).

CREATE  dHDR1  dHDR1.LEN  ALLOT

: OLDHDR1  ( -- )  dHDR1  dHDR1.LEN  ERASE ;
```

```
Screen # 26
( READ dBASE FILE 1                              14:41 04/30/87 )

HANDLE  F1              :  ?F1  F1  .HDR ;

: SET.F1.NAME  ( -- )  F1  FILENAME  WFDREP25.DBF ;

: PREPARE.F1  ( -- )
        SET.F1.NAME  F1  OPEN.FILE  F1  dHDR1  PREPARE ;
```

```
Screen # 27
( BUFFER FOR dBASE FILE HEADERS                  14:41 04/30/87 )

15000  CONSTANT  dHDR2.LEN

CREATE  dHDR2  dHDR2.LEN  ALLOT

: OLDHDR2  ( -- )  dHDR2  dHDR2.LEN  ERASE ;

HANDLE  F2              :  ?F2  F2  .HDR ;

: SET.F2.NAME  ( -- )  F2  FILENAME  WFDREP55.DBF ;

: PREPARE.F2  ( -- )
        SET.F2.NAME  F2  OPEN.FILE  F2  dHDR2  PREPARE ;
```

```
Screen # 31
( WORDS FOR OUTPUT BUFFER                        15:54 03/30/87 )

: WALL ;
7       CONSTANT  IN$LEN
7  DUP  CONSTANT  #FYS
9  DUP  CONSTANT  OUT$LEN  +
4  DUP  CONSTANT  OUT.ID.LEN  +
        CONSTANT  OUT.REC.LEN

500  OUT.REC.LEN  *  CONSTANT  OUT.LEN

CREATE  OUT.BUF  OUT.LEN  ALLOT  : OB  OUT.BUF ;
\  Make sure OUT.BUF is large enough  e.g. #F1 recs  + #F2 recs

: CLOB  ( -- )  OUT.BUF  OUT.LEN  BLANK ;
```

```
Screen # 32
( DATA FOR OUTPUT BUFFER                         15:54 03/30/87

VARIABLE  DELTA
VARIABLE  FLD.CT

: OUT.ID  ( rec# -- addr)  OUT.REC.LEN *  OUT.BUF + ;
                           \ Execution address is used
CREATE  'OUT.ID    ' OUT.ID  ,  \ word ID under MATCH.FOUND
                           \ MATCH.FOUND? ( addr -- n
: SET.'ID  ( -- )  'OUT.ID @  'ID ! ;

: ID>OUT.BUF  ( rec#in rec#out -- )
        SWAP  dREC  SWAP  ID  4  CMOVE ;

: ID>PRN  ( rec# -- )  dREC  CR  4  TYPE ;
```

```
Screen # 33
( ADD UNMATCHED RECORDS FROM PROF BUDG        13:31 04/20/87 )

VARIABLE  #RECS
VARIABLE  #NEW.RECS

: $$  ( rec$ field$ -- addr )
       SWAP  ID  @ +  SWAP  DELTA @  -  OUT$LEN @  + ;

: $$>OUT.BUF  ( rec$in rec#out -- )  FLD.CT @ 1+  2  ?DO  OVER
     I  @FIELD  1+  OVER  I  $$  IN$LEN  CMOVE  LOOP  2DROP ;

: NEW.REC.CT  ( -- n)  #RECS @  #NEW.RECS @  + ;

: NEW.REC>OUT.BUF  ( line# -- )
       NEW.REC.CT  2DUP  ID>OUT.BUF  $$>OUT.BUF ;
```

```
Screen # 34
( ID FOR OUTPUT BUFFER                        10:58 03/30/87 )

: LOAD.CURR.BUDGET  ( -- )
       2 DELTA !  #FLDS  FLD.CT !  REC.CT W@  #RECS !
       REC.CT W@  0  ?DO
       I  DUP  ID>OUT.BUF  1  DUP  $$>OUT.BUF  LOOP ;

: LOAD.PROF.BUDG  ( -- )
       0 DELTA !  #FLDS  FLD.CT !  0 #NEW.RECS !
       REC.CT W@  0  ?DO
       I  DUP  @REC  MATCH.FOUND?  IF  $$>OUT.BUF
       ELSE  DROP  NEW.REC>OUT.BUF  1 #NEW.RECS +!
                                          THEN  LOOP ;
```

```
Screen # 35
( GENERATE DATA FILE FROM 2 INPUT FILES       17:01 03/30/87 )

CLOE
SET. II

' PREPARE.F1' COUNT  TY
       PREPARE.F1
' LOAD.CURR.BUDGET' COUNT  TY
       REC.CT W@  CT !  LOAD.CURR.BUDGET  \  CT used by word
' PREPARE.F2' COUNT  TY                   \  MATCH.FOUND?
       PREPARE.F2
' LOAD.PROF.BUDG' COUNT  TY
       LOAD.PROF.BUDG
\ This sequence generates a buffer of combined data

MID.BEEP
```

```
Screen # 36


HANDLE OBF    : ?OBF  OBF  .HCE ;
OBF  FILENAME  COSTA.DAT
OBF  MAKE-FILE
OBF  OUT.REC.LEN  NEW.REC.CT  *  OUT.F
OBF  CLF
```

(Continued from page 30.)

versal, and are a real hindrance to the adoption of Forth. Anyone uploading a file has a moral obligation to define every non-standard word. Since F83 is well known and available without cost, it can serve as a secondary standard. —Leonard

>>>> New File <<<<
Name: VIDEO1.ARC Number: 911
Address: LMORGENSTERN
Bytes: 11340 Library: 3

Description: This is a complete video driver, written by Bill Beers, to which I have added an F83 prelude and short documentation. Keywords: VIDEO, MS/DOS, MS-DOS, BEERS, F83, 8086, 8088, GRAPHICS, DISPLAY, SOURCE

Category 3, Topic 1 Message 27
Sat Feb 06, 1988 M.HAWLEY

Thanks, Leonard. The prelude and VIDEO blocks loaded and ran on the first try. I especially enjoyed your explanation of your thought processes and methods of deciphering someone else's code. This was a valuable lesson to a novice like myself. Now I'm sorting through the various words in the VIDEO block and trying to learn how to use them.

FORML

IMPROVED STRING HANDLING

MIKE ELOLA - SAN JOSE, CALIFORNIA

At the kernel level, Forth string handling can stand improvement. Through better factoring, these kernel enhancements should result in a kernel at least as small as existing Forth kernels.

Opportunities remain for better kernel factoring, for at least three reasons:

1. There are kernel words which perform multiple functions (FIND and WORD are two examples), so they have too large a functional scope.
2. There are kernel words with overlapping scope, suggesting that their scope is not sufficiently unique. This kind of factoring defect has led to a proliferation of in-line string operators which function similarly.
3. Even more incremental factoring of properly-scoped words is possible. This could lead to a reduction of the kernel size as the need for kernel versions of certain words is eliminated.

WORD is one example of a kernel word that contains multiple functions. It parses a word or phrase, leaving another copy of it atop the dictionary in the form of a counted string. Perhaps the parsing function can be factored in upcoming standards, to make it separately available. *[Write the proposal, Mike. -Ed.]*

For maximum flexibility, the parsing function must not be overblown and must have as few prerequisites as possible. Currently, WORD is inflexible because it parses text only from the currently selected input stream, be it either the text-input buffer (TIB) or a block. Better parsing functions would not assume a given input location, or even that a copy of the parsed expression need be made and placed elsewhere. As will be shown, less ambitious parsing functions can be framed.

While the scope of SPACES cannot be criticized, SPACES could still be derived more incrementally. If SPACES is used rarely, then a short phrase based upon a more general primitive may serve as well as SPACES. (Likewise, SPACE could be replaced by the phrase 1 SPACES.)

Avoiding reiteration of similar code is desirable outside of the kernel, as well as inside it. When creating counterparts to SPACES, such as CRS, UNDERLINES, and BACKSPACES, repeated use is made of DO-LOOP phrases. However, an even more general primitive could be made part of the supplied, minimal Forth. Programmers could add any higher-level spinoffs, as desired (page 18, Figure One-a). Then, for example, SPACES could be defined in terms of EMIT-CYCLES (Figure One-b).

A second form of this definition is shown within parentheses. To compile an ASCII character code into a definition, the word ASCII is usually added to Forth. It rids the dictionary of constants like BL, COLON, UNDERLINE, etc. However, because ASCII cannot parse a space, BL cannot be eliminated.

As a counterpoint of ASCII, C@" cannot parse a quote character. But the character code for a space is needed more often, and C@" parses the space character properly. Also, since C@" ignores any extra characters, a clearly delineated comment can be included in the parsed string (Figure Two).

The definition of C@" and another, unconventional form of the definition are shown in Figure Three.

As a step towards explaining, let's consider the compilation of ASCII control codes. Both ASCII and C@" are inadequate when it comes to parsing control characters with the legibility needed in source code. Using constants alleviates the situation with respect to control characters, but name choices like CR already have been taken. Here I would suggest constants named with the C@" prefix and quote suffix. In the examples shown in Figure Four, the quoted string is part of the name field. An alternate naming convention using ASCII-prefixed words would require even longer names.

The addition of C@"2QUOTE" is convenient, as was evidenced in the parenthesized definition of C@". The addition of ASCII-SPACE (or the inconsistent BL) should likewise be added for a group of words prefixed by ASCII.

Using either C@" or ASCII, character codes are compiled as in-line integers. At run time, the in-line integer is placed on the stack by (LIT), the word uniquely reserved for this function. This is proper factoring. However, in the case of in-line strings, the equivalent function has been reiterated over and over again.

An author who has exposed these problems and has offered solutions is Don Colburn ("A Consistent Structure for In-Line String Literals," *FORML Conference Proceedings*, 1982). He has observed and corrected the overlapping functionality of the following words: (."), ("), and (ABORT"). He also addresses the problem of placing a table of strings into the dictionary. Currently, the kernel only supports string compilation within a definition via the immediate words .", ", and ABORT".

Colburn has developed a string compil-

ing word that works outside of a colon definition and that forms the basis for a group of well-factored routines. This baseline word works equally well within a definition when it is called by the group of immediate words mentioned in the last paragraph.

(Moreover, Colburn's high-level definition of `(.")` enables it to reference `TYPE`. Sometimes, functional consistency should be the driving force behind our choice of factoring options. By revectoring `EMIT`, one reasonably expects that all output will be affected, since `EMIT` is thought to be the one portal through which all output text must travel. However, because `(.")` may be defined without reference to either `EMIT` or `TYPE`, its behavior may remain unchanged after the revectoring of `EMIT` has been performed. If the standard would require Colburn's kind of implementation, then revectoring `EMIT` would be certain to affect all output similarly.)

Criticism of Colburn's code can be answered with minor changes. For instance, to eliminate manipulation of the return stack, `($LIT)` could be implemented as a `CODE` word.

Among other possible changes are a couple that reflect my own wishes. Rather than parse up to a quote character, I want to compile a string in the dictionary delimited however I choose (similar to `WORD`). I also prefer a more general-sounding name for this function. A definition that would meet these added objectives (although the internals of Colburn's algorithm may be more universally workable than this one) is offered in Figure Five. Colburn's baseline definition can then be defined as in Figure Six.

Discrete words for parsing and comparing strings should be part of the Forth kernel, and these could become components of `WORD` and `FIND`. To keep the kernel streamlined, only basic versions of each need be adopted (similar to the policy suggested for `EMIT-CYCLES`).

For string comparisons, one baseline word could be used as the platform for other spinoffs, including any structure-specific comparison operators. Such an approach would avoid dictating the use of a particular type of string structure.

So the design needed is one that would obtain required input parameters from the stack instead of directly from memory. This is the way `TYPE` accesses its length

parameter in order to remain as functional with uncounted strings as it is with strings in a counted format. On page 33 of their book *Forth Tools and Applications*, Gary Feierbach and Paul Thomas provide a solution with these attributes. A slightly modified version of their routine is shown in Figure Seven.

As mentioned at the start, several string-parsing routines will be explored. Any of these could be used as the basis for other spinoffs, as desired.

To include a primitive parsing function in the kernel that does not favor one representation of strings over another, a function with less scope than WORD is required.

The definition in Figure Eight-a could be used as a baseline parsing routine. The variable SCAN-INCREMENT determines the direction of scanning. If it is -1, then scanning is to be performed from high memory towards low memory. If it is 1, then scanning is from low memory towards high memory. If the search span length is exhausted without finding the delimiter, then the top-of-stack (TOS) returned is zero, and below it is the search character rather than the address where the delimiter resides.

A version that would be able to return the first character that is not a delimiter character (to suppress leading delimiters), could be developed in a similar fashion (Figure Eight-b).

To fold both routines into one requires another variable. If SCAN-GOAL is set to zero, then the scan continues until a non-delimiter is found. If SCAN-GOAL is set to a Boolean truth value (-1 or 1, depending on your system), then the scan continues until a delimiter is found (Figure Eight-c).

Conclusions

Kernel string operations encompass a great deal of functionality. Regrettably, all of these individual functions are not in a form that can be readily used. A significant advantage of Forth is the reusability of modularly coded routines. However, where modularly factored kernel functions are lacking, this strategy is uprooted.

This article has been excerpted from a book-length manuscript in progress. I am still seeking writeups of string packages that can be reproduced and analyzed alongside other string packages presented in the book. Mail submissions to me in care of FIG.

```
: EMIT-CYCLES  ( #repetitions char - - )
  OVER 0> 0= IF 2DROP EXIT THEN
  SWAP 0 DO
    DUP EMIT
  LOOP DROP ;
```
Figure One-a.

```
: SPACES  ( #spaces - - )
  BL EMIT-CYCLES ;
    ( C@"   " EMIT-CYCLES ; )
```
Figure One-b.

```
C@" Quit" OVER = IF QUIT THEN
C@" Main" OVER = IF MAIN-MENU THEN
C@" ?help" OVER = IF HELP THEN
```
Figure Two.

```
: C@"
  34 WORD 1+ C@
  [COMPILE] LITERAL ; IMMEDIATE
```

unconventional:

```
  C@"2QUOTE" WORD 1+ C@
  [COMPILE] LITERAL ; IMMEDIATE
```
Figure Three.

```
 13 CONSTANT C@"CR"
 12 CONSTANT C@"FORMFEED"
  8 CONSTANT C@"BACKSPACE"
  7 CONSTANT C@"BELL"
... CONSTANT C@"F1" ...F9
... CONSTANT C@"LEFT"
... CONSTANT C@"RIGHT"
... CONSTANT C@"UP"
... CONSTANT C@"DOWN"
127 CONSTANT C@"DELETE"
```
Figure Four.

```
: WORDS,  ( <string> ( char - - )
  WORD C@ 1+ ALLOT ;
```
Figure Five.

```
: ," ( <string>" ( - - )
  ASCII " WORDS, ;
  ( C@"2QUOTE" WORDS, ; )
```
Figure Six.

```
VARIABLE COMPARE-BUF
: COMPARE            ( addr len addr - - -1/0/1 )
  0 COMPARE-BUF !    ( assume equality )
  SWAP 0 DO          ( addr addr - - )
  OVER C@ OVER C@ - ?DUP
  IF 0< IF 1 ELSE -1 THEN
    COMPARE-BUF ! LEAVE
  ELSE 1+ SWAP 1+ SWAP THEN
  LOOP               ( addr addr - - )
  2DROP  COMPARE-BUF @ ;
```
Figure Seven.

```
VARIABLE SCAN-INCREMENT
: BI-CSCAN  ( addr span char - - addr+- rem.span )
  SWAP ROT  ( char span addr - - )
  DDUP +    ( char span addr addr.end - - )
  SCAN-INCREMENT @
  0> IF SWAP ELSE 1- >R 1- >R THEN
  DO   ( char span - - )
    OVER I C@ = IF ( match: )
      I ROT DROP SWAP LEAVE
    ELSE
      1-  ( char span- - - ) THEN
  SCAN-INCREMENT @ /LOOP
```
Figure Eight-a.

```
: -BI-SCAN   ( addr span char - - addr+- rem.span )
  SWAP ROT   ( char span addr - - )
  DDUP +     ( char span addr addr.end - - )
  SCAN-INCREMENT @
  0> IF SWAP ELSE 1- >R 1- R> THEN
  DO   ( char span - - )
    OVER I C@ = 0= IF ( no match: )
      I ROT DROP SWAP LEAVE
    ELSE
    1-  ( char span- - - ) THEN
  SCAN-INCREMENT @ /LOOP ;
```
Figure Eight-b.

```
VARIABLE SCAN-GOAL
: /BI-CSCAN   ( addr span char - - addr+- rem.span )
  SWAP ROT    ( char span addr - - )
  DDUP +      ( char span addr addr.end - - )
  SCAN-INCREMENT @
  0> IF SWAP ELSE 1- >R 1- R> THEN
  DO   ( char span - - )
    OVER I C@ = SCAN-GOAL @ = IF
      I ROT DROP SWAP LEAVE
    ELSE
      1-  ( char span- - - ) THEN
  SCAN-INCREMENT @ /LOOP ;
```
Figure Eight-c.

F83

HEADERLESS LOCAL VARIABLES AND CONSTANTS

JOHN P. DAUGHERTY - BURKE, VIRGINIA

A current application under design is large, and dictionary space has become a premium. For readability and speed, the coding reflects heavy use of 'local' variables and constants within word components, eliminating the slower and less readable DUPs, ROTs, and SWAPs. Some method of stripping off the headers of these local variables and constants would save measurable space. Researching existing literature (see references) turned up unsatisfactory methods. Instead, a new design was chosen and is discussed below. Laxen and Perry's F83 is used.

Backgroup and Requirements

Real headerless local variables and constants should execute the same as their standard counterparts created with VARIABLE and CONSTANT. Four bytes in any variable or constant word are key to execution: the two bytes of the code field address (CFA) and the two bytes containing the variable or constant. The header is not needed for execution and can be discarded. Therefore, any method for defining local variables or constants must retain only four bytes.

To meet the application needs, the following requirements were established:

1. During execution, the local variables and constants must perform identically to the standard, global variables and constants.
2. An application must be able to name the local variable or constant, like any Forth word. Numbers are not permitted.
3. The defining words for the local variables and constants shall be VARIABLE and CONSTANT. This minimizes changes to existing source code.
4. Both local and global variables and constants will be used. Therefore, an application must be able to pick the correct defining word, either the global or the local one.
5. When all components are finished using the local words, the local headers must be stripped and any memory used by them recovered. The link in their headers must be somehow taken care of to allow normal dictionary searches to occur after headers are stripped.

Approach

By placing the view, link, and name fields (the header) in free memory space several thousand bytes above the current dictionary pointer, and by placing the code and parameter fields in the normal dictionary space, headerless variables and constants are possible. Once finished using these local variables and constants, their headers can be forgotten by unlinking them from the other in-line Forth words. The space used by these headers will be overwritten by the dictionary as it grows. To make sure the standard and local defining words are distinguishable, a separate vocabulary will be used. The dictionary search order will control which defining word is found first.

Screens 1 - 4 contain the utility that meets the requirements outlined above. All but three of the components in this utility are in the HIDDEN vocabulary. LOCAL-OFFSET, START-LOCAL, and END-LOCAL are in the FORTH vocabulary.

START-LOCAL (screen 4) saves the CURRENT vocabulary threads (four in F83) after setting HIDDEN as the first vocabulary to search in the dictionary. Saving these threads becomes important when unlinking the local words with END-LOCAL. LOCDP, the local dictionary pointer, is initialized to high memory (determined by the constant LOCAL-OFFSET and HERE). This distance allows sufficient room for most applications, and can be changed as necessary with F83's IS (e.g., 3000 IS LOCAL-OFFSET).

New versions of CREATE, CONSTANT, VARIABLE, and 2VARIABLE are defined in screen 3. These words are HIDDEN definitions, distinguishing them from the Forth definitions of the same name. DOLOCAL is used directly or indirectly by each of these words. It creates a header in high memory by setting the dictionary pointer, DP, to the value contained in LOCDP after saving DP's value on the stack. With DP in high memory, any new words CREATEd will have headers well above the normal dictionary space. This is okay as long as this space is not overwritten before it is unlinked.

After creating the header in high memory, the original value of DP is compiled (DUP ,) into the space following the header to be retrieved later (by DOES>@). The CFA of the local variable or constant will be placed at the address pointed to by DOES> @. This will be the indirectly threaded address stored in any parameter field using the defined local word. Words defined with DOLOCAL are made IMMEDIATE to allow DOES> to present the CFA address for compilation in a colon component, or to be EXECUTEd when not compiling. STATE determines which of these actions to take.

Wrapping up the CREATE portion of DOLOCAL, the LOCDP is updated to point

just beyond the last high memory used, ready for the next local header. Having finished with the high memory, DP is reset to the normal dictionary space and a value is compiled with , (comma). In the case of CONSTANT, the word to "comma" is the headerless Forth word DOCONSTANT, found by ticking the constant zero. Likewise, for VARIABLE — DOCREATE, another Forth headerless word, is compiled.

Words defined by these four defining words, when used inside a colon definition, perform in the same way as Forth definitions. No speed penalty is sacrificed by using local words. Additional compile time is required, but this can be traded against fewer words in the dictionary to search, once the local words are unlinked.

Unlinking is accomplished by END-LOCAL (screen 4). For each thread, END-LOCAL searches for any links to memory above the current dictionary pointer. If one is found, a false flag is left on the stack and also TUCKed below it. BEGIN-WHILE-REPEAT terminates and the IF, seeing a false flag, branches to ELSE and stores the saved THREAD in the word linked to high memory, thus unlinking this particular thread from the local constant or variable. If no link in high memory is found, the BEGIN-WHILE-REPEAT loop will terminate when the link points to a value below what is stored in THREADS. A true flag will be on the stack, and IF drops the two addresses. END-LOCAL also resets the vocabulary search order to where it was originally.

Rules of Use

1. Place all global variables and constants together, ahead of the local variables and constants.
2. Begin all local variables with START-LOCAL.
3. Do not change vocabularies or FORGET any words between START-LOCAL and END-LOCAL. Doing so will change the links, and END-LOCAL will not unlink properly. A system crash is likely!
4. Debug the program before making words local. DEBUG doesn't like headerless words.
5. Make sure the program is not larger than LOCAL-OFFSET. If it is, change LOCAL-OFFSET's value before loading.
6. Use END-LOCAL when finished using the local words in other words.

Summary

This utility encourages use of local constants and variables. It meets all the requirements outlined above. As long as the rules are followed, no problems have been noticed to date. It has almost eliminated the confusing code associated with overuse of DUPs, ROTs, and SWAPs. A tradeoff in compilation speed is not noticeable, and the run-time code is unaffected. It is not foolproof if the rules are not followed.

References

1. Greene, Ronald L. "A Proposal for Implementing Local Words in Forth," *The Journal of Forth Application and Research*, Vol. 2 No. 4, 1984, pp. 39-42.
2. Morgenstern, Leonard. "Anonymous Variables," *Forth Dimensions*, Vol. IV No. 1, May/June 1984, pp. 33-34.
3. Pruitt, Carol. "Local Definitions," *Forth Dimensions*, Vol. VI No. 6, March/April 1985, pp. 16-17.

ADVERTISERS INDEX

```
     0
 0
 1
 2          **********************
 3             Real Headerless
 4                Variables
 5                   and
 6                Constants
 7          **********************
 8
 9
10
11
12 J. Patrick Daugherty
13 6403 Four Oaks Lane
14 Burke, Va   22015
15
```

```
     1
 0 \ Load Screen
 1
 2 only forth also definitions
 3
 4 fence off  forget empty
 5
 6 warning off
 7 1 3 +thru
 8 warning on
 9
10 only forth also definitions
11
12 mark empty  here fence !
13
14
15
```

```
     2
 0 \ Constants and Variables
 1
 2
 3 2000 constant local-offset
 4
 5 only forth also hidden also definitions
 6
 7 variable locdo
 8
 9 create threads  #threads 2* allot
10
11
12
13
14
15
```

```
     3
19Jul86jpd \ Hidden words                                    19Jul86jpd

: dolocal  ( adr  -- )
           here locdo @  do !
   create  duo ,       immediate
           here 'locdo ' do ! ,
   does)   @ state @  if , else execute then ;

: create    ( -- )    [ ' scr @ ] literal   dolocal    :
: constant  ( n -- )  [ ' 0 @ ]   literal   dolocal ,  ;

: variable  ( -- )    create   0 , ;
: 2variable ( -- )    variable 0 , ;

only forth also definitions hidden also
```

```
     4
19Jul86jpd \ Start- and End-local                            19Jul86jpd

: start-local ( -- )
   context @  avoc !  hidden          ( search hidden first)
   current @  threads #threads 2* move
   here  local-offset +   locdo !  ;

: end-local ( -- )
   avoc @ context !   #threads 0
   do   threads i 2* + @   current @ i 2* +  ( adr1 adr2 )
        begin   2dup  @ u(  over @
                here u(  tuck and      ( adr1 adr2  flag1 flag2)
        while   drop @
        repeat
        if 2drop  else  ! then
   loop ;
```

```
     5
19Jul86jpd \ Example from Thinking Forth ( modified)         19Jul86jpd

200 constant max-height                  \ global
500 constant max-width

start-local     variable right    variable left
                variable top      variable bottom
: box ( left, top, right, bottom -- )
   bottom !  right !  top !  left !
   top @  bottom @  -  max-height )  abort" Too High!"
   right @ left  @  -  max-width  )  abort" Too Wide!"
   left  @  top @     right @  top @      line
   right @  top @     right @  bottom @   line
   right @  bottom @  left @   bottom @   line
   left  @  bottom @  left @   top    @   line  ;
end-local
```

HAVE YOUR ASSEMBLER...

DARRYL C. OLIVIER - NEW ORLEANS, LOUISIANA

...and eat it, too. One of the great virtues of Forth is its compactness. We can pack a lot of program into a small memory space. Another advantage is its on-line assembler, which gives our programs blinding speed and complete control of the hardware. These two might seem to conflict. An assembler takes up a lot of dictionary space. However, this need not be the case.

The assembler is only needed at compile time when a CODE word is encountered. It is not needed at all when the program runs. Consider the structure of a definition. All Forth words have a code field that contains the address of machine-executable code. When the word is executed, the computer jumps to the address pointed to by the code field, and executes the run-time code contained there. For the typical Forth word, this run-time code is contained in its defining word and is common to all words so defined. For this reason, the defining word must exist when the daughter word is executed.

The code field of an assembler definition, on the other hand, points not to some run-time code common to all assembler definitions, but to its own parameter field which contains machine-executable code. For this reason, the assembler definition is independent of its defining word.

It is very simple to use the assembler at compile time but to eliminate it from the compiled program. The strategy is to load your Forth kernel as usual, then change the dictionary pointer to some arbitrary location in high memory. Load the assembler, then return the dictionary pointer to its old value. Load the rest of the application, being sure that all CODE definitions are loaded early, before the assembler is overwritten. The vocabulary mechanism ensures that Forth will be able to find its way around this fragmented dictionary. (See Figures One-a, One-b, and One-c.)

In the example shown in Figure Two, DP is the variable that contains the dictionary pointer.

Using this technique, you can have an assembler that is just as big and fancy as you like, with no increase in the size of your program.

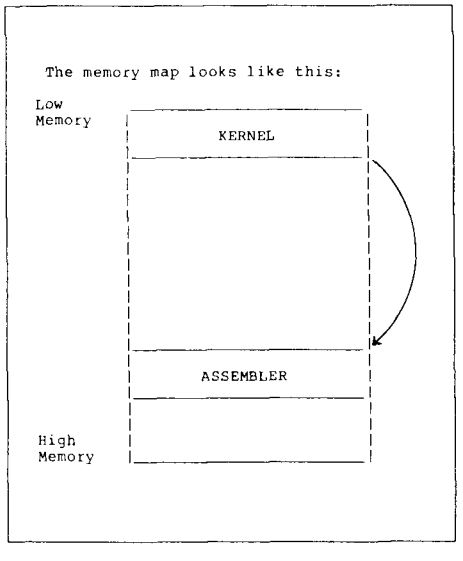


Figure One-a

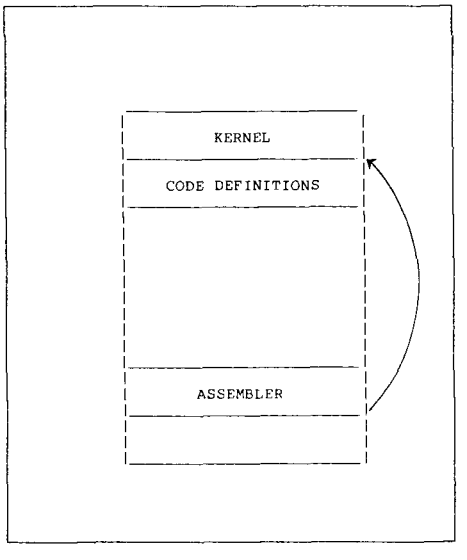


Figure One-b

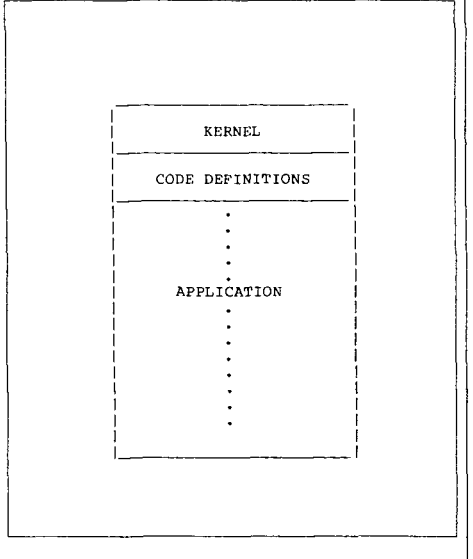


Figure One-c

```
VOCABULARY  ASSEMBLER  IMMEDIATE

HERE            \ The current value of the dictionary
                \ pointer remains on the stack while the
                \ assembler is loaded.

A000  DP !      \ Changes the dictionary pointer to an
                \ arbitrary value in high memory.
                \ (40k in this example)
                        .
                        .
                        .
                The assembler is loaded.
                        .
                        .
                        .
DP !    \ Returns the dictionary pointer to the old
        \ value left on the stack by HERE.

FORTH DEFINITIONS
                        .
                        .
                        .
    The rest of the application is loaded.
```

Figure Two

FEBRUARY 1988

ANS FORTH MEETING NOTES

RAY DUNCAN - MARINA DEL REY, CALIFORNIA

The third meeting of the X3J14 ANS Forth Technical Committee was held at Redondo Beach, California on February 10 - 12, 1988. FORTH, Inc. hosted the meeting, providing us with nice facilities, refreshments, copying, administrative services, and even restaurant recommendations! The weather was clear and balmy, with temperatures in the 80s, and the TC members were also treated to a small earthquake.

As of the beginning of this meeting, the X3 Secretariat had not yet appointed all the permanent officers for X3J14. In the meantime, Elizabeth Rather agreed to continue as Acting Chair, Martin Tracy as Acting Secretary, and Ron Braithwaite as Document Editor. I have been appointed as Vice-Chair by X3J14, and the position of International Representative is still vacant.

Approximately 25 people attended this meeting, which I believe makes it the largest to date. There were several observers from the Los Angeles area and several new members, including George Shaw (Shaw Laboratories, Inc.) and Dean Sanderson (FORTH, Inc.). Guy Kelly was present for the first day but had to leave early after delivering a report from the Research Subcommittee; two other valued members from previous meetings, Jerry Shifrin and Larry Forsley, were unable to attend at all due to other work obligations.

Approximately thirty technical proposals were received prior to, or at, the third TC meeting. The TC's Technical Subcommittee found time to consider nearly all of these, under the excellent guidance of Greg Bailey.

First, a brief review of the Standards process so far. At its first meeting, the X3J14 Technical Committee adopted the text of the Forth-83 Standard (less the Experimental Proposals) as its working, or BASIS, document. The BASIS will evolve into the draft proposed American National Standard (dpANS) document, as the TC ratifies Technical Proposals which delete, amend, or enlarge the BASIS.

Prior to this meeting, the most significant change made to BASIS was Elizabeth Rather's proposal to (in essence) relax Forth-83's insistence on 16-bit stacks and addresses, and to allow the construction of Standard systems based on virtual Forth machines with other word sizes. The Document Committee also recast the BASIS from its original form into a style consistent with ANSI requirements.

The most important Technical Proposals passed at this meeting were (these are paraphrased by me and are not the original wording, unless otherwise noted):

1. To continue the process of decoupling the Standard from machine word size by substituting the terms "single precision" and "double precision" for "16-bit" and "32-bit."

2. To require any Standard Program to include a list of resource requirements, System Implementation Options, and environmental dependencies. System Implementation Options are defined as features, properties, parameters, boundary conditions, behaviors, or side effects in which respect Standard Systems are permitted to differ (this includes word size).

3. To relax the definition and/or requirement of Equivalent Execution in order to allow for Standard Systems implemented on different physical or virtual machines. It seemed to be the prevailing opinion of the TC (which I share) that Forth-83's definition of Equivalent Execution is a marginally useful concept at best. It can never be guaranteed unless the supporting hardware and software are in fact identical, which is hardly the situation the Standard is intended to address.

4. To remove the requirement for floored division from the Standard, while still making it available for existing Forth-83 Standard programs or other programs which require its rounding behavior. The exact wording of this Technical Proposal follows:

Add to the 'Definitions' section the following:

"Division: Division produces a quotient 'q' and a remainder 'r' by dividing operand 'a' by operand 'b'. A division operation may return either 'q', 'r', or both.

"The identity 'bq + r = a' shall hold for all 'a' and 'b'.

"When both integers are divided and the division is inexact, if both 'a' and 'b' are positive, then 'q' is the largest integer less than the true quotient and 'r' is positive. If either operand is negative, whether 'q' is the largest integer less than the true quotient ('flooring') or the smallest integer greater than the true quotient ('rounding down') is implementation-defined, as is the sign of 'r'.

"An implementation shall clearly document which negative-case behavior is the default, and shall provide a clearly documented mechanism whereby the other behavior will be available to the programmer."

5. `NIP` and `TUCK` were added to the Controlled Reference Word Set, in recog-

nition of their widespread usage.

6. EVAL, a word that allows the interpretation of arbitrary strings, was added to the BASIS. This word only originated with Martin Tracy about a year ago, but has been rapidly accepted in the Forth community. When EVAL is available, words that redirect the input stream can be readily ported from one system to another, and new interpreters can be easily built in an implementation-independent manner.

7. The Documentation Committee was directed to add Appendices to BASIS that will (1) identify problem areas in program portability and (2) provide a rationale for each difference between ANS FORTH and Forth-83. These Appendices presently exist only as "stubs," but will be developed over subsequent meetings.

While not passed during this meeting, a Technical Proposal for a minimum extension set of floating-point operators enjoyed widespread support within the TC. The Technical Proposal was referred to Martin Tracy for fine-tuning; he was directed to confer with those vendors currently supplying floating-point implementations and make sure that it did not contain any major conflicts with existing practice. After any necessary "adjustments," the Technical Proposal will be reconsidered at the next X3J14 meeting.

The TC demonstrated its concern for compatibility with the existing Forth-83 Standard by rejecting Technical Proposals that would have changed the meaning of NOT and removed the requirement that ." , .(, (, and ABORT" be able to parse null strings. It also defeated a proposal to rename 2OVER, 2SWAP, et al. to DOVER, DSWAP, etc.

The BASIS document (in its current state) is available for the cost of reproduction and postage from:

Secretary
ANSI ASC X3/X3J14 Forth Standards Committee
111 N. Sepulveda Blvd.
Suite 300
Manhattan Beach, CA 90266

Technical Proposal forms and instructions are also available from the Secretary.

Warning: While the X3J14 BASIS document is public, it is not to be reproduced or distributed in any way without the express permission of the X3J14 Technical Committee. BASIS is not, and never will be, a Standard. It is only an early precursor of dpANS, a work-in-progress of X3J14. BASIS is very fluid; nothing in it can be counted on to be in the eventual dpANS. Note also that the dpANS, once completed, will be subject to an extensive process of public review and possible revision before it becomes an American National Standard for Programming Language Forth.

The next meeting of X3J14 will be in Rochester, New York on May 11 - 14, 1988. New members and observers are welcome, Technical Proposals from all interested parties are invited. To be placed on the mailing list for the next meeting, address your request to:

Chair, X3J14 Forth Standards Committee at the address above.

Personal impressions: While it got off to a slow start at the first two meetings, the X3J14 TC has now demonstrated a desire, and ability, to move briskly forward on some very difficult issues. In spite of their vastly different constituencies and historical perspectives, the TC members worked together productively in a spirit of cooperation and goodwill. I was particularly impressed (and surprised) by the TC's willingness to embrace a relatively new but remarkably useful tool-building word such as EVAL, and by the high degree of interest in extension word sets for file interfaces and floating point.

The X3J14 members represent an extraordinary assemblage of Forth expertise. They represent the cutting edge of Forth technology in virtually every area: from Forth "engines," to "rich" Forth implementations for workstations such as the Mac II, to NASA's Massively Parallel Processor. I am convinced that the ANS Forth Standard, when it is finished, will represent an immense leap forward for Forth vendors and programmers.

This is an unofficial report of the X3J14 meeting and does not constitute official minutes. Any opinions, predictions, or conclusions in this report are purely my own and should not be misconstrued to represent those of the X3J14 TC or its other officers.

CALL FOR PAPERS

for the tenth annual

FORML CONFERENCE

The original technical conference
for professional Forth programmers, managers, vendors, and users.

Following Thanksgiving, November 25–27, 1988

Asilomar Conference Center
Monterey Peninsula overlooking the Pacific Ocean
Pacific Grove, California U.S.A.

Theme: Forth and Artificial Intelligence

Artificial intelligence applications are currently showing great promise when developers focus on easy-to-use software that doesn't require specialized expensive computers. Forth's design allows programmers to modify the Forth language to support the unique needs of artificial intelligence. Papers are invited that address relevant issues such as:

Programming tools for AI
Multiusers and multitasking
Management of large memory spaces
Meeting customer needs with Forth AI programs
Windowing, menu driven or command line systems
Captive Forth systems—operating under an OS
Interfacing with other languages
Transportability of AI programs
Forth in hardware for AI
System security

Papers about other Forth topics are also welcome. Mail your abstract(s) of 100 words or less by September 1, 1988 to:

FORML
P. O. Box 8231
San Jose, CA 95155

Completed papers are due by October 15, 1988. For registration information call the Forth Interest Group business office at (408) 277-0668 or write to **FORML.**

Asilomar is a wonderful place for a conference. It combines comfortable meeting and living accommodations with secluded forests on a Pacific Ocean beach. Registration includes deluxe rooms, all meals, and nightly wine and cheese parties.

March 1988

THE BEST OF GENIE

GARY SMITH - LITTLE ROCK, ARKANSAS

News from the GEnie Forth RoundTable. When I was asked to begin a "Best of GEnie" column for *Forth Dimensions*, I immediately realized I wanted Leonard Morgenstern's topic area to be among the first presented. Leonard has done a marvelous job of presenting our language of choice in bite-sized chunks, and in such an entertaining fashion as to create a novice-and-experienced Forth tutorial area.

Time constraints and the sheer depth of material available in Leonard's area dictated making this first column an extraction of that one topic. Future installations will include discussions of the ANS FORTH X3J14 Technical Committee. Those of you who are not aware that the future of Forth is being decided on GEnie *now* are missing an opportunity to make your wishes known.

Besides serving as the centerpiece in the first "Best of GEnie," Leonard had several coinciding events that make March 1988 especially memorable. He just celebrated a birthday and his retirement from the rat race. He also graciously accepted an offer to join Dennis Ruffer, Scott Squires, and me as co-hosts of the Forth RoundTable on GEnie. Congratulations, Leonard, and thank you.

Category 3, Topic 1 Message 3
Sat Jan 16, 1988 M.HAWLEY

Dennis, The word OK? appears in the ANAGRAMS.ARC file I recently downloaded from this board. It was in another that I can't remember now. ANAGRAMS looked like a lot of fun but it wouldn't run on any of my public-domain Forths because of the missing OK?. As a newcomer to Forth, I find this sort of thing very frustrating. I thought this was the purpose of having standards. Another problem word I ran into was 2ARRAY. It was in BULLCOWS for ZENFORTH but not defined in the version of ZENFORTH that I have. As an example of how it should be done. I downloaded UNARC.SCR and UNARC.F83 which is a prelude for F83 to run this application written in some non-standard Forth, NORTHFORTH or something like that. It worked perfectly the first time. My compliments to that masterful programmer. As a beginner, I run into dead ends real fast when I hit an undefined word which is not in the Forth-83 wordset or any of the manuals I have. Thanks for your attention to my problems. I imagine others have had similar experiences and get turned off for that reason. —meh

Category 3, Topic 1 Message 5
Sun Jan 17, 1988 S.W.SQUIRES [scott]

STANDARDS PEOPLE, please note message #3 above. Finding undefined words in source code is a pet peeve of mine. As a beginner, I went through the same problem and sometimes still do. It's no fun typing in a tool you've been wanting and then finding that it is missing one word with no info as to what it even does. It's also very easy to assume that "everyone" has this word and knows what it is (2ARRAY, ASCII, MYSELF, etc.) Now is 2ARRAY a two-dimensional-array-defining word or a single-dimension, double-precision array? I've seen it used both ways.

This is where the standards come in — common words should be in the standard, or at least in the controlled reference word set. A utility that would come in handy would be the ability to compare words with the raw standards and display the ones that aren't in the standards and that aren't defined in the standards. The Forth-79 from MicroMotion used to do this by setting one of the unused bits in the name field if the word was standard. —Scott

Category 3, Topic 1 Message 6
Sun Jan 17, 1988 M.HAWLEY

Dennis, Let me correct myself. The hangup word in ANAGRAMS was NUF? I have since found a definition of NUF? in F83X320C ... a version of Forth-83 that I don't usually use.

```
: NUF?
KEY? DUP IF 2DROP
BEEP KEY 13 = THEN ;
```

The hangup word 2ARRAY appears in LIFE.SCR — the game of life contained here in LIFE.ARC. OK?, I believe, appeared in BULLCOWS written for ZENFORTH and I'm vaguely aware that there is a newer version of ZEN than the one I have. Last, but not least, I was unable to load MAZE.BLK from MAZE.ARC because of the word VIDEO. Is VIDEO a standard Forth-83 word? I don't find it in the book *Inside F83* by C. H. Ting. And I can't find it in my public-domain Forths including F83, F83x320C, ZENFORTH, SEATTLE FORTH, MVP-FORTH. This gets old fast. It would be nice to be able to assume that an application uploaded here for public consumption would contain definitions for all non-F83 standard words, but this does not seem to be the case. Another correction, the definition of NUF? above omitted one KEY. It should be:

```
: NUF?
KEY? DUP IF KEY 2DROP
BEEP KEY 13 = THEN ;
```
—meh

Category 3, Topic 1 Message 7
Sun Jan 17, 1988 LMORGENSTERN [leonard]

To M.HAWLEY and Scott Squires: I sympathize with anyone who has run into the problem of trying to load a utility and finding undefined words in it. There is no other name for it than bad documentation. Sometimes it is due to laziness or sloppiness, but more often, I think, it is inexperience. Forth makes it possible for green programmers to do very clever and useful things. Not knowing better, they assume that others can use their source code. (There wasn't any difficulty with BASIC, was there?) If this is so, new and better standards, although much needed in other ways, will not solve this problem; we already have the resources for it.

There are two solutions, at least one of which should always be applied by the programmer who submits a file: One is to meticulously compare the words in his/her source with the prelude. The other solution is to load the prelude and application on top of some widely available version of Forth, such as F83 or even fig-FORTH. If it loads and runs without flaw, an experienced programmer will know how to modify it to run on another system.

Journals, such as *Forth Dimensions*, always check this sort of thing before publication. However, we don't have the resources for that kind of thing on a bulletin board like this one, and have to trust those who submit files. Unfortunately, it is a matter of Let the User Beware!

Category 3, Topic 1 Message 10
Tue Jan 19, 1988 J.SHIFRIN

Leonard, I'm afraid you're incorrect about journals like *Forth Dimensions* checking to make sure that all published code was standard. In fact, that was one of my greatest frustrations in getting started with Forth. *FD* seems to publish about any dialect (fig-FORTH, Forth-79, Forth-83, special implementations) without regard for portability. It typically includes a fair amount of CODE definitions as well. Personally, I gave up on getting useful code from *FD*, other than ideas and approaches.

Category 3, Topic 1 Message 12
Tue Jan 19, 1988 JAX [Jack Woehr]

In defense of *Forth Dimensions*: Actually, the assumption there is that you have *every* issue from way back when, where these "nonstandard" words like RECURSE were first mentioned. It's just added motivation to buy bound volumes.

Category 3, Topic 1 Message 13
Wed Jan 20, 1988 J.W.BAXTER

As another defense of *Forth Dimensions*, they do mark each listing as to what sort of Forth it is. If they made some arbitrary choice (all submissions must be Forth-83, for example), they would have even fewer submissions to choose from. I think they're on the right track. —John

Category 3, Topic 1 Message 23
Mon Jan 25, 1988 LMORGENSTERN

To M.HAWLEY: To write pixels in Forth, you have to access the DOS video driver or use the ANSI system. The exact call depends on your system. In MS-DOS you would use a call to interrupt 10H. There is a file 660 VIDEO.ARC that is described as "Video words for IBM F83." Look into it. It would be appreciated if you would let us know whether it does what you want. — Leonard.

Category 3, Topic 1 Message 25
Fri Jan 29, 1988 M.HAWLEY

Well, VIDEO.ARC looks promising. I unarced it with UNARC.F83, a good example of a useful application that runs using only standard words (and F83 in particular). The unarced file is VIDEO.BLK, which looks similar to the VIDEO.BLK file from VIDEOF83.ARC. Unfortunately, it did not load completely. It hangs on the word SLL, which I don't find in the standard. At first I thought it was an assembler word, but it is not in a code definition. So, frustration again.

I'm just an amateur programmer who wants to learn Forth for the sheer fun of it. I am persuaded that Forth is the most versatile and, in some sense, the most powerful of languages—potentially. But I am still singing the Forth-Beginner's Blues.

To: M.HAWLEY

I downloaded VIDEO.ARC and found your problem line, which is line 13 of screen 2. It reads:

```
: pg>bh
```

```
page# @ 8 sll ; (s--bx )
```

You are right that SLL is not in the standard — it is not even in F83. I have worked out what it does, illustrating the value of well-chosen word names and good documentation. We start with the stack diagram, contained within the "stack parentheses," preferred by some programmers over ordinary parentheses. The diagram says that PG>BH leaves BX on the stack. In Intel 8086/88 assembler, BX is the whole B register, and BH is its high-order part. Now, the documentation for certain 10H interrupts requires that BH contain the page number.

Next, we look at the word name PG>BH. Clearly, it puts the video page number into BH. It is now a short step to recognizing that SLL shifts BX left by 8 bits, discarding whatever is pushed off the high end, and filling the low end with zero bits.

Now comes the "Aha!" SLL is "Shift Left Logical," easy enough to write in F83 assembler. Continuing, I find that some additional words need definition:

#SPLIT (n - - hi lo)
Split a word into two bytes. Note: Most Forths have the stack diagram (n - - lo hi)

#JOIN (hi lo - - n)
Opposite of #SPLIT. Same comments as for #SPLIT.

C@L (seg ofs - - b) "Long C-fetch."
Similar to C@, but the stack specifies a segment and offset. Most Forths use (ofs seg - - b).

C!L (b seg ofs - -) "Long C-store."
Same comments as for C@L.

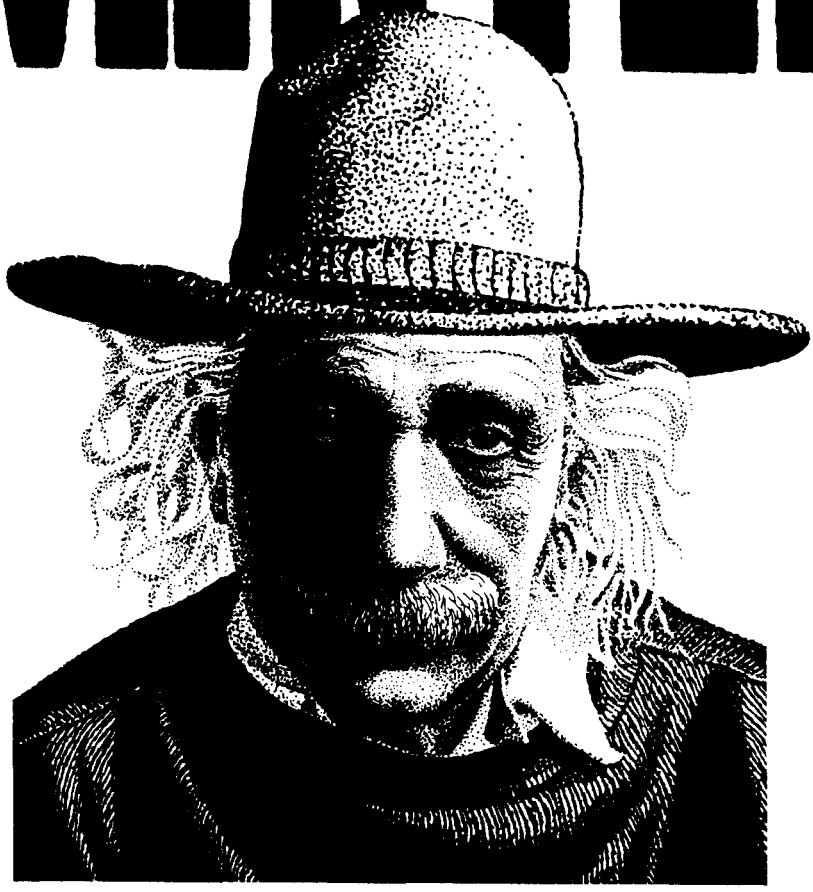
CMOVEL (ofs1 seg1 ofs2 seg2 n - -) "Long CMOVE"
Move data from one segment to another. Same comments.

I have written a prelude named F83PRELU.BLK, and have combined it with VIDEO.BLK and this note, into a new VIDEO.ARC file. If the sysops accept it, it will replace the previous VIDEO.ARC. I will upload it to you personally, so you will have it regardless.

The problems you are having are uni-

(Continued on page 14.)

WANTED



$1000 REWARD

for the **World's Fastest Programmer.**

Be the first to put our mystery gizmo through its paces and win $1000! Use any computer, any software. The showdown is at the **Real-Time Programming Convention**, Nov 18-19th, Anaheim, Calif. For complete rules, write:
Programming Contest, Forth Interest Group, PO Box 8231, San Jose, CA 95155.

Guide to "ANSI X3J14 Forth Technical Proposal"

X3J14 is an American National Standards Institute (ANSI) Technical Committee developing a standard for the Forth programming language.

We want your input. Please read this to learn more about the standards effort and how to get your comments to the committee.

X3 is the part of ANSI that deals with Information Processing Systems. J14 is a Technical Committee sanctioned by X3 to develop a standard for Forth. Our charter is to recommend a "draft proposed American National Standard" (dpANS) to X3. X3 in turn will recommend the dpANS to ANSI. If, based on public review, the dpANS meets ANSI requirements for technical accuracy and consensus support the dpANS will become the American National Standard Forth.

Our goal is to complete the dpANS by November 1988. While that date may sound far away, your input is needed now.

We have adopted sections 1-16 of the FORTH-83 standard as a starting point for the dpANS. Please use it as a reference point for your comments. We ask that you keep these guidelines in mind:

1. The standard is to be evolutionary - not revolutionary. First and foremost, X3J14 is intent on codifying existing common practice (words like DUP, SWAP, DROP). We are also working to resolve conflicts when more than one common practice exists. Finally, we are considering areas not currently covered by standards, like floating point. We are not going to restructure Forth.

2. Be constructive. If you don't like a facet of FORTH-83, please propose something better.

3. Proposals must be typewritten. You need not use the standard form, but please provide the same information in the same general layout. Detailed instructions are provided below. If possible, in addition to a paper copy of your proposal, please send a copy on diskette. Straight ASCII text file, MS-DOS or Macintosh format, 3.5" or 5.25" diskette.

4. Send proposals to the address shown on the proposal form. Please enclose a self-addressed, stamped envelope.

Additional information on ANSI, the standards process, and X3J14 is available from

X3 Secretariat/CBEMA
311 First Street, NW
Suite 500
Washington, DC 20001-2178

Documents of interest:

X3/SD-0 Information Brochure
X3/SD-2 Organization and Procedures
X3/SD-5 Standards Criteria
X3J14/87-005 Proposal for ANS Forth
X3J14/87-002 Draft Scope of Work
X3J14/87-003 Draft Plan of Work

X3J14/87-021 12/04/87

Filling out the "ANSI X3J14 Forth Technical Proposal" form

Page:	Please number each page of your proposal.
Title:	A short phrase that characterizes your proposal. Example: String extensions.
Related Proposals:	If you have submitted other proposals on this subject, please list their titles and dates.
Proposal ( ) Comment ( )	Are you proposing a specific change? Then check the Proposal box. Otherwise, check the comment box.
Keyword(s):	Pick a keyword that helps others understand what area of Forth your proposal addresses. For example: Kernel, double integers, system word set...
Forth word(s):	List all affected Forth words, including ones added or deleted by your proposal or discussed in your comments.
Abstract:	Briefly convey the nature of your proposal (or comment) in broad terms.
Proposal:	State your proposal in specific terms. When proposing a new word, or changes to an existing word, simply state the new definition. Include a stack picture if appropriate.
Discussion:	If you are making a comment, put it here. If you are submitting a proposal, provide compelling arguments in favor of your proposal. Be concise. If you are aware of arguments against your proposal, state them and rebut them. Is your proposal consistent with current Forth conventions? Does the proposal involve hardware or other system dependancies? Does it affect application portability? Does it have implications for multi-user environments? Should the proposed change be mandatory? Does it affect ROMability? Is it general purpose? How does it affect execution speed? How does it affect compilation speed? What are its memory requirements?
Submitted by:	Provide your name, address, and daytime phone number.

Use the "ANSI X3J14 Technical Proposal Form, cont'd" form if additional pages are needed. Remember to put page numbers and repeat the "Submitted by" and "Date" fields on each page.

X3J14/87-021 12/04/87

ANSI X3J14 Forth Technical Proposal	Page: of
Title:	
Related Proposals:	
Keyword(s):	Proposal ( ) Comment ( )
Forth Word(s):	
Abstract:	
Proposal:	
Discussion:	
Submitted by: Address: Phone:	Date:
ANSI X3J14 Forth Standards Committee 111 N. Sepulveda Blvd., Suite 300, Manhattan Beach, CA 90266	

X3J14/87-021 12/04/87

ANSI X3J14 Forth Technical Proposal, cont'd	Page: of
Title:	

Submitted by: Date:
Address:

Phone:

ANSI X3J14 Forth Standards Committee
111 N. Sepulveda Blvd., Suite 300, Manhattan Beach, CA 90266

X3J14/87-021 12/04/87