$10 Volume XIV, Number 2 July 1992 August

FORTH DIMENSIONS

Interfacing to XMS

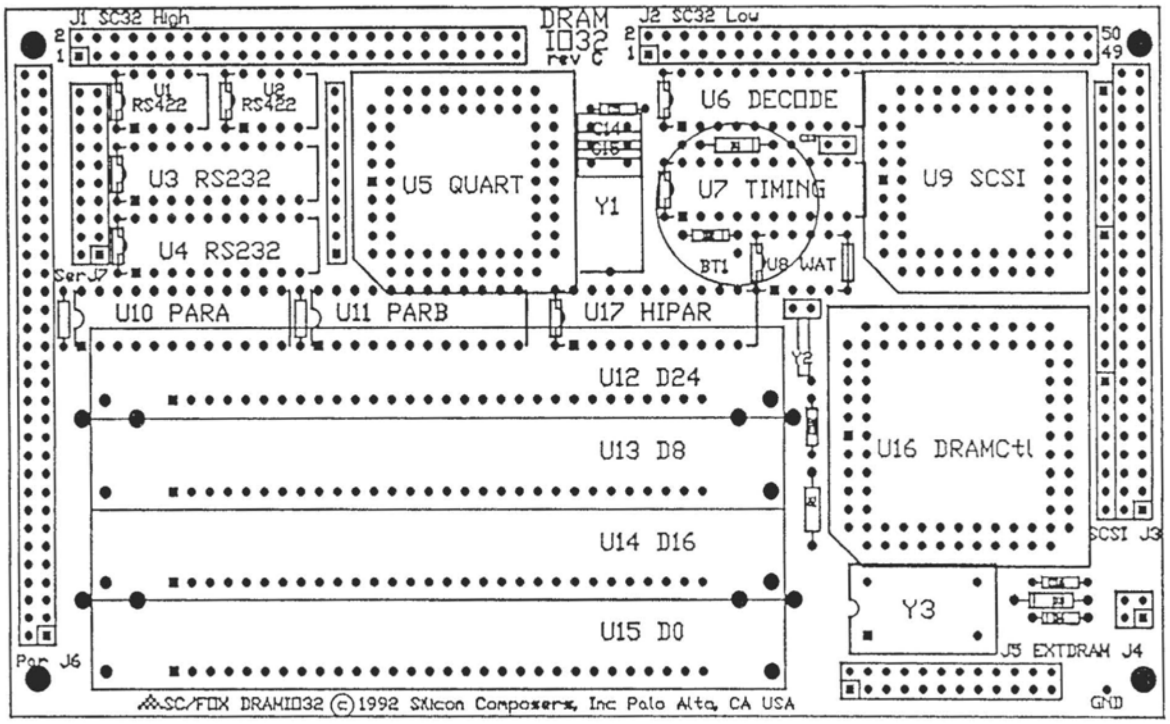
Random Disk Records

Structured Pattern Matching

ANS Forth Update

SILICON COMPOSERS INC

Announcing the SC/FOX DRAMIO32 Board



SC/FOX DRAMIO32 Board (actual size)

- The DRAMIO32 is a plug-on daughter board which attaches directly to either the SBC32 stand-alone or PCS32 PC plug-in single board computers.
- Up to 16 MB on-board DRAM.
- 5 MB/sec SCSI controller supports up to 7 SCSI devices.
- 16-bit bidirectional parallel port, may be configured as two 8-bit ports.
- 4 Serial ports, configurable as 4 RS232 or 2 RS232 and 2 RS422.
- Each serial port is separately programmable in 33 standard baud rates up to 230K baud.
- 4 input handshaking and 6 output control lines.
- 7 general purpose latched TTL level output lines.
- 11 general purpose TTL level input lines with interrupts available on either transition.
- 2 programmable counter/timers, may use internal or external event trigger and/or time base.
- Wristwatch chip keeps correct time and date (battery included) with or without system power.
- 24 bytes of keep-alive CMOS RAM, powered by wristwatch battery.
- Source code driver software and test routines for SCSI, parallel and serial ports, DRAM, timers, CMOS RAM and wristwatch chip included.
- Interrupts available for all I/O devices.
- No jumpers, totally software configurable.
- Hardware support for fast parallel to SCSI transfer.
- Multiple boards may be stacked in one system.
- Two 50-pin user application connectors.
- Single +5 Volt low-power operation.
- Full power and ground planes.
- Input for external +5 volt supply to keep DRAM data in case of loss of main power.
- 6 layer, Eurocard-size: 100mm x 160mm.
- User manual and interface schematics included.

See application article in this issue.
For additional product and pricing information, please contact us at:
SILICON COMPOSERS INC 208 California Avenue, Palo Alto, CA 94306 (415) 322-8763

Contents

Editorial

Last Chance

There is still time to enter our contest for articles about "Forth on a Grand Scale." Many people still aren't aware of large Forth applications, of Forth components in complex systems, of significant multi-programmer projects, etc. To improve the general awareness of how Forth performs in "the big leagues," and to enlighten those who only know the lean and mean side of Forth, we are offering cash awards to the top three papers we receive on the subject. See the ad on page 40 of this issue for details—better do it now, since the contest deadline is August 3!

Of course, articles received after that date will still be considered for publication, they just won't qualify for the prize money.

Speaking of Which...

Forth Dimensions thrives best when its readers participate most. We are constantly looking for a broad range of articles: applications, utilities, tutorials, vendor/developer interviews and success stories, examples of Forth technique, hardware projects, essays, and letters to the editor come to mind. Because we are such a widely distributed and diverse community, it is important for you to remember that we are indeed interested in what *you* are doing. Not every article should be very advanced (our "hot thermometer" rating)—many readers need more moderate fare at their current stage of Forth expertise. So don't feel, because your *magnum opus* isn't forthcoming, that you have nothing to contribute. We want to hear from you!

Update your Rolodex

Those of you who have followed the Forth Interest Group for any length of time know that, like any organization, it is always seeking to improve its efficiency. As part of the most recent changes, please note the new address and telephone numbers on our masthead and mail-order form. The change in offices will, it is believed, substantially help consolidate and coordinate the business of running this membership organization.

—Marlin Ouverson

ANS Forth Update

From Elizabeth D. Rather, chairperson of the committee (X3J14) developing ANS Forth, we received this report dated 3/23/92:

"The four-month public review period for the Draft Proposed ANS Forth (dpANS) closed February 25, 1992. X3J14, the Technical Committee developing ANS Forth, received a total of 34 official comments and three late comments. Many of the comments were multi-part, producing a total of over 200 discrete items. Of the 37 comments, only three were negative. The others were generally supportive, pointing out unclear issues, making specific suggestions, and noting typos.

"All the public review comments, as well as a number of technical proposals, were considered by X3J14 in its 19th meeting held March 17–21 at Athena Programming near Portland, Oregon. Reported typos will be fixed, many of the suggestions were adopted (or alternative solutions were found for the problems raised), and the document was clarified in many areas.

"As required by X3 procedures, all commentors will receive responses drafted and approved by the TC. In addition, the TC agreed to send each commentor the complete set of comments and responses, and a copy of the changes made to the document.

"The TC was pleased at the amount of time and effort invested in the comments. Many interesting and valid points were raised, and the document is much improved. The improvements are incremental in nature, with no radical changes in policy or approach (except for the three comments noted above, extensive revisions were not requested). Several words were dropped from extension wordsets, and some others were added (particularly in the Floating Point wordset). A few words got more-mnemonic names, descriptions and requirements were clarified, and more Rationale material was added.

"The revised dpANS will be released for a two-month public review soon, probably early or mid-May. As before, copies will be available from Global Engineering Documents, 2805 McGaw Avenue, Irvine, California 92714 (800-854-7179, or 714-261-1455 from outside the U.S.A. and Canada). Comments received during this period will be considered in September. This process will repeat until the TC determines that no additional substantive changes need to be made, at which time the document will be forwarded to X3 for processing as an ANS."

Forth Dimensions
Volume XIV, Number 2
July-August 1992

Published by the
Forth Interest Group

Editor
Marlin Ouverson

Circulation/Order Desk
Frank Hall

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

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

The Forth Interest Group

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

"*Forth Dimensions* (ISSN 0884-0822) is published bimonthly for $40/46/52 per year by the Forth Interest Group, 1330 S. Bascom Ave., Suite D, San Jose, CA 95128. Second-class postage paid at San Jose, CA. POSTMASTER: Send address changes to *Forth Dimensions*, P.O. Box 2154, Oakland, CA 94621."

Letters

Letters to the Editor—and to your fellow readers—are always welcome. Respond to articles, describe your latest projects, ask for input, advise the Forth community, or simply share a recent insight. Code is also welcome, but is optional. Letters may be edited for clarity and length. We want to hear from you!

Response and Apology

Herein I hope to apologize for my errors, explain myself more fully, and suggest some further articles I'd like to see published.

I would like to apologize to anyone who took comments in my article in issue XIII/2 as a criticism of Jim Callahan's HS/FORTH. It was certainly not my intention to imply his Forth was anything other than a fine product. Although I have not used it personally, I have never heard a bad word about HS/FORTH or its author.

My comments were, however, intended to be a criticism of a claim in his advertisement. While merely my own opinion, and others may read his ad differently, I felt his ad's claim of compiling 40,000 lines per minute was almost meaningless, in that no details were given as to how the figure was reached. I tried to illustrate, using some specific timings on Pygmy, that it makes a hell of a difference whether you are running on a 4.77 MHz PC or a 33 MHz '386. In his letter in issue XIII/4, he supplied some of those details (running on a '286, etc.). I think his ad would be stronger by including the processor and clock speed details but, of course, that is entirely his business. Indeed, I appreciate his advertising in *Forth Dimensions* and thereby helping to support it. My exceptions to certain parts of his ad should be kept in perspective.

Jim also suggested *Forth Dimensions* has an anti-vendor bias. If it does, I have not been aware of it. It is my belief my article on Pygmy was published because it was one of the best available at the time, rather than because *Forth Dimensions* is biased against vendors. I would be most happy to see *Forth Dimensions* publish an article written by Jim about his HS/FORTH. And, if I am wrong about the anti-vendor bias, I would like to see Jim write a letter or article putting forth his evidence and reasoning.

I am also struck by another aspect of Jim's advertisement: it seems to imply some of HS/FORTH has been pirated by others and/or that some of his development work has been stolen or misused. My first reaction is to reject this as a vague claim given without supporting details. If there are supporting details, I think a full treatment of this subject would make a great article, and I would like to encourage Jim to write it!
Sincerely,
Frank Sergeant
809 W. San Antonio St.
San Marcos, Texas 78666

Threading a Memory Waste

Dear Sir:

I would like to thank Guy Kelly for producing an article reviewing a few aspects of so many Forth systems [*FD* XIII/6]. Such articles are long overdue in *Forth Dimensions* and will prove a great service to anyone interested in Forth.

I would, however, like to expand upon a few of the points covered. The main emphasis of the article was on the relative speeds of the various systems and architectures, and is therefore incomplete without an analysis of the costs associated with any speed advantage. True optimization involves economy in the use of memory, and the avoidance of built-in limits to expansion, as well as raw execution performance. We examined both direct and subroutine threading a decade ago—and rejected both for reasons that obviously still hold today. They waste memory for marginal speed improvement, better obtained where needed by local optimization. When attempts are made to limit the memory wasted, the inevitable result is unacceptable limits on program space available and sacrifice of the performance gained.

Direct threading, while marginally faster than indirect (for very simple programs) uses anywhere from six to 20 extra bytes for every single colon definition. Paragraph-aligned list structures such as used by F-PC waste an additional average of eight bytes per colon definition. Memory and disk space may be cheap these days, but they are always limited, especially if you aren't your only client.

Speaking of waste, some Forths such as polyFORTH which claim multi-segment program space do so by duplicating the entire base of the application in each segment so that a three-segment, 172K program has the lower 32K of each of the three segments identical, wasting 64K of core and disk. HS/FORTH is multi-segment without any duplication or waste.

Clearly, an optimizer that totally eliminates `NEXT` linkage where speed is essential will produce code that is faster and overall more compact than direct threading. We were curious how the LMI optimizer seemed to give a faster Sieve time than ours. Mystery solved. They optimize variables and `DO LOOP` indices as embedded constants and hope they'll never change. Works fine for the Sieve, good luck in real life.
Sincerely,
Jim Callahan
Harvard Softworks
P.O. Box 69
Springboro, Ohio 45066

Optimizer Concern Misplaced

Dear Marlin:

Congratulations on publishing the fine article by Guy Kelly, and thank you for forwarding the copy of Mr. Callahan's letter. Fortunately, Mr. Callahan's concern about the applicability of our optimizer to real-life situations is misplaced. The optimizer specifically looks for situations where `DO LOOP` indices and other parameters are in fact constants, as is the case with the `DO LOOP` indices in the Sieve benchmark, and then compiles the values as embedded literals. When the parameters for a `DO LOOP` are not constants—for example, if they are passed into a definition on the stack or loaded from variables—the optimizer generates completely different code.
Regards,
Ray Duncan
Laboratory Microsystems Inc.
12555 W. Jefferson Blvd., Suite 202
Los Angeles, California 90066

(Continued on page 42.)

Interfacing to Extended Memory

Jesus Consuegra
Sitges, Spain

This paper describes a practical approach to interface to the XMS (Extended Memory Specification), to allow MSDOS Forth programs to access the memory beyond the artificial limit of 640 Kbytes that MSDOS can directly handle.

Introduction

After many years digging around and trying many different Forth dialects, last year I found the Forth environment of my dreams: UDFORTH, a compact yet powerful implementation of Forth-83, with some extensions and only one incompatibility. The product is manufactured by a company called Upper Deck Systems. The product is quite robust, with an easy interface to the user: it uses standard text files and includes a full-screen, pull-down-menus text editor that drops you automatically into an offending error when it occurs. Although the manual is a little sparse, a Forth programmer can easily find his way through this system with the help of the sample programs included (a reduced version of the Unix utility grep among them).

This interface was born as a requirement to hold big streams of MIDI data from an electronic musical instrument.

One of the big criticisms of Forth is that one has to reinvent the wheel for him/herself any time one wishes to use a specific tool or feature. The unavailability of compiled code to link to Forth, and the large number of different Forth dialects in use has prevented wider use of Forth in commercial environments.

As an example of how one has to set up their own tools, I'm going to describe an XMS (Extended Memory Specification) interface for MSDOS.

After some wars on expanding or extending memory, Microsoft decided to establish an unified extended memory specification for 80286/386/486 class machines running MSDOS. The availability of big chunks of memory can dramatically improve the performance of many programs.

This interface was born as a requirement to hold big streams of MIDI data from an electronic musical instrument but, since then, I've used it in other business programs to hold large arrays of data without the need to first define virtual disks in memory and then treat the arrays as if they were disk files.

The first thing a programmer has to do to interface to XMS is find out where the service routine sits in memory. To do so, one has to issue an interrupt $2F, that tells if XMS is or is not installed. After that, another call to int $2F with a different parameter gives back the address of the service routine. Once the address of XMS is known, calling the different services is straightforward: just place a specific value at registers AX and DX and issue a far call to the service routine.

This seems (and is) simple, but at the time of writing the program, I had no idea about how to issue such a call—even in machine code.

After some unsuccessful querying on CompuServe (I have no access to GEnie from Spain), a friend of mine (Juanma Barranquero, co-sysop of the Forth conference at NEXUS) and a couple of hours in front of a cold meal, gave me the clue (and the basics of 808X machine code I needed) to do it. The result is a two-line sequence of Forth-assembly instructions:

```
' xms_address >body # bp mov
  es: 0 [bp] far call
```

with `xms_address` being a 2variable.

Once this barrier was surpassed, the rest of the interface came in a worknight. That's the power of Forth!

The code that comes with this paper is a self-explanatory sample of some of the functions supplied by the XMS driver. It should be easy to port to other Forth implementations.

Currently I'm working on another interface, this time to the Btrieve Record Manager. And in the queue, a CUA-compliant, text-only, full-featured windowing interface is waiting.

With those tools in hand, to write business programs in Forth will be considerably easier. Upper Deck Systems has promised me a Windows version of UDFORTH. When it becomes available, I'll stick to Forth for all my programming.

Code of XMSTEST.S

```
\ MODULE Xms

\ This module has been written by Jesus Consuegra, who places it in the
\ public domain. Although the author has reasonably checked the code,
\ there are no implicit or explicit warranties that this code does not
\ contain errors or mistakes. The users take all responsibility to check
\ whether it is suitable for their applications. Use of this code is at
\ your own risk. The author will appreciate comments, suggestions and
\ any kind of communication that can improve the wide spread of FORTH.
\
\ Mail address:     Jesus Consuegra
\                   C/Enric Morera 36, esc 3, 2-2
\                   Edifici Les Neus
\                   08870-SITGES
\                   (SPAIN)
\
\ Bix:              jesusc
\ Compuserve:       100014,3112
\ EUNET:            jesusc@met.foro.es
```

```
\ =======================================
\ Some stuff to neatly print hex figures
\ =======================================

: (2hex)
     9 > if
            ascii A + 10 -
     else
            ascii 0 +
     then  ;

: 2hex.
     dup 0< if
            256 +
     then
     dup
   16 / dup      \ print high byte in hex
     (2hex)
     emit
   16 mod dup    \ print low byte in hex
     (2hex)
     emit  ;

: 4hex.
     dup 8 shr
     2hex.
     $ff and
     2hex.  ;

: 4hex:.
     dup 8 shr
   2hex. ." :"
     $ff and
   2hex.  ;
```

```
\ =======================================
\ Tools for managing regs
\ =======================================

: regAH
    regAX $ff00 and 8 shr  ;

: regAL
    regAX $ff and  ;

: regBH
    regBX $ff00 and 8 shr  ;

: regBL
    regBX $ff and  ;

\ =======================================
\ The XMS interface
\ =======================================

: xms?  ( --- f )
\ Print error message and return a flag
  $4300 !> regAX
  $2f int86
  regAL $80 = if
        ." XMS  installed "     true
        else
        ." XMS not installed. " false cr
        then  ;

2variable xms_address

: get_xms_driver_address ( --- )
\ regES:regBX are the address
    $4310 !> regAX
        $2f int86
        regES regBX xms_address 2!   ;

: Show_Xms_Driver_Address ( --- )
    get_xms_driver_address
    ." at address: "
    regES 4hex. regBX ." :" 4hex. cr  ;

\ ==========================================
\ Generic XMS request
\ ==========================================

code xms_req ( --- )
    ds push
    bx push
    bp push

    ' regAX >body # bx mov
    es: 0 [bx] ax mov
    ' regDX >body # bx mov
    es: 0 [bx] dx mov

\  Call XMS service

      ' xms_address >body # bp mov
    es: 0 [bp] far call

\  Return values to standard regS

      ax es: ' regAX >body #) mov
      bx es: ' regBX >body #) mov
      dx es: ' regDX >body #) mov

    bp pop
      bx pop
    ds pop
      next
end-code

0 value Ermes

: E$! !> Ermes ;

: ApiError ( Ercode --- )
  case
   $80  of $" Function not implemented."                E$! endof
   $81  of $" VDISK device detected."                   E$! endof
   $82  of $" A20 error."                               E$! endof
   $8e  of $" General Driver error."                    E$! endof
   $8f  of $" Unrecoverable driver error."              E$! endof
   $90  of $" HMA does not exist."                      E$! endof
   $91  of $" HMA already in use."                      E$! endof
   $92  of $" DX is less than /HMAMIN=."                E$! endof
   $93  of $" HMA not allocated."                       E$! endof
   $94  of $" A20 line is still active."                E$! endof
   $a0  of $" All extended memory is allocated."        E$! endof
   $a1  of $" All extended memory handles are in use."  E$! endof
   $a2  of $" Invalid handle."                          E$! endof
   $a3  of $" Invalid Source Handle."                   E$! endof
```

Forth Interest Group
Statement of Change in Financial Position
April 30, 1990 to April 30, 1991

	4/30/90	4/30/91	Change + = Increase - = Decrease
ASSETS:			
Current Assets:			
Foothill Bank, Money Market	15,925.38	24,865.91	8,940.53
Foothill Bank, Checking	636.43	700.14	63.71
Pending Foreign Clearing	-102.00	51.67	153.67
Returned Checks Pending	0.00	72.00	72.00
FORML, Money Market	15,894.11	16,916.46	1,022.35
FORML, Checking	1,926.01	1,236.64	-689.37
Total Current Assets:	34,279.93	43,842.82	9,562.89
Inventory:			
Inventory at cost	34,147.53	26,601.17	-7,546.36
Total Inventory:	34,147.53	26,601.17	-7,546.36
Other Assets:			
Deposit, United Parcel Service	200.00	200.00	0.00
Second Class Postal Account	156.31	192.41	36.10
Accounts Receivable	3,166.20	2,099.00	-1,067.20
Total Other Assets:	3,522.51	2,491.41	-1,031.10
TOTAL ASSETS:	71,949.97	72,935.40	985.43
LIABILITIES:			
Sales Tax	33.89	35.58	1.69
FD Dues alloc. to future months	37,487.30	41,518.51	4,031.21
TOTAL LIABILITIES:	37,521.19	41,554.09	4,032.90
Financial Reserve:	34,428.78	31,381.31	-3,047.47

```
   $a4  of $" Invalid Source offset."                  E$! endof
   $a5  of $" Invalid Destination handle."             E$! endof
   $a6  of $" Invalid Destination Offset."             E$! endof
   $a7  of $" Invalid length."                         E$! endof
   $a8  of $" Invalid overlap in move."                E$! endof
   $a9  of $" Parity error."                           E$! endof
   $aa  of $" Block not locked."                       E$! endof
   $ab  of $" Block is locked."                        E$! endof
   $ac  of $" Block lock count overflow."              E$! endof
   $ad  of $" Lock failed."                            E$! endof
   $b0  of $" A smaller UMB is available."             E$! endof
   $b1  of $" No UMBs available."                      E$! endof
   $b2  of $" Invalid UMB segment number."             E$! endof
\
           $" Unknown error code."                     E$!
  endcase
  ." Xms: " Ermes $.  ;

: Show_Xms_Version
    $ffff !> regDX
        0 !> regAX
    xms_req
    regAX ." Version: " 4hex:. cr
    regBX ." Internal driver revision: "
          4hex:. cr
    regDX 1 = if
    ." HMA does exist."
    else
    ." HMA doesn't exist."
    then cr  ;

: Show_HMA
    $ffff !> regDX
     $100 !> regAX
    xms_req
    regAX 1 = if
          ." HMA assigned to the caller."
      $200 !> regAX
          xms_req               \ free HMA
            regAX 1 = if
            ." HMA successfully released"
            else
            ." Error on releasing HMA" cr
            regBL ApiError
            then
            cr
      else
      regBL ApiError
      then
      cr  ;

: Show_A20_Line_Status
    $700 !> regAX
    xms_req
      regAX 1 = if
      ." A20 line is physically enabled."
      else
      regBL ApiError
```

```
      then
      cr  ;

: Query_free_Exmem
      $0800 !> regAX
      xms_req
      regBL 0<>  if
            regBL ApiError
      else
      ." Largest free ext.memory block is "
      regAX u. ." Kbytes." cr
      ." Total free extended memory is "
      regDX u. ." Kbytes."
      then
      cr  ;

: DoTest
      Show_Xms_Driver_Address
      Show_Xms_Version
      Show_HMA
      Show_A20_Line_Status
      Query_free_Exmem  ;

: XmsTest
    cls
      xms? if DoTest
      then  ;

\ if you have the shareware version only,
\ you have to do a save-exe instead.

turnkey XmsTest XMSTEST.EXE

\ End of file
windowing
```

Random Disk Records

Brian Sutton
Tampa, Florida

A number of years ago, I began writing programs to use in my chiropractic practice to try to make life a little easier. Since my computer was rather limited (it was a TI-99/4a), Forth was naturally the only available language that offered even a glimmer of hope of turning out something useful in a reasonable length of time.

Ten years and a number of computers later, I think I'm starting to make some progress. The random access words described in this article form the basis of my accounting program that tracks patient accounts, bills insurance, keeps the checkbook balanced, and generates profit-and-loss statements, among other things.

For a while, I put up with many bothersome housekeeping tasks that programs often have to perform, such as opening all the data files when the program starts, closing them all upon termination of the program, making sure the data gets saved out just before the program terminates, etc. Another major consideration was lack of RAM (32K in the TI and 64K in CP/M) and how to deal with the large amounts of data I needed to process.

Thankfully, the F83 buffer system caught my attention and seemed to be a simple and effective answer to my problems. If the virtual memory concept worked so well for Forth screens, why not use it for data too?

So these words are basically an adaptation of the Forth screen buffers, with a few extra features thrown in. The major ways these tools make programming much easier are:

- The data files are automatically opened when needed. If the file isn't used, it isn't opened.
- As data is entered and the file fills, it is automatically increased in size to accommodate the new data. You can specify the size of each increment of file growth—I use the number of bytes in a cluster on my hard disk (8K). You can also specify that the program asks your permission before appending any file.
- The buffers are of the F83, truly-least-recently-used scheme. This means that if you use four buffers, you can have four different data addresses in memory without worrying about accessing an address that suddenly contains a different piece of data.
- Also, each updated record is saved to disk before the buffer is re-used for something else. This is real handy here in Florida, where power outages are often a daily occurrence during the rainy season. The most we ever lose is the last entry in the journal, if that.
- Unlike the F83 buffers, you can mix records of different sizes in the buffers. For example, one record might be 256 bytes while another is two. Even though my buffers are 1024 bytes, if a two-byte record is called only two bytes are read or written.
- Each record is referenced just like a variable or an array. Entering its name returns an address on the stack that you can use to fetch or store text or values. Substitute the word `ACCOUNT_NAME` for the Forth word `BLOCK` to see how this works. While `10 BLOCK` returns the starting address of a 1024-byte Forth screen, `10 ACCOUNT_NAME` might return the starting address of the 24-byte name of account number ten.
- The error-handling routines display which record or file was being accessed when the error occurred.
- The word `FLUSH` saves all the records to disk and then closes all the record files. This is especially useful if you use a lot of data files—DOS won't let you have more that 15 open handles (in addition to the console, printer, etc.) per program, even though the total number of open system files can be much higher. FCBs can be used to get around this limit (yuck!) or you can write your own handle table, but I prefer to have my program execute `FLUSH` every time it returns to a main menu. This offers additional data security and solves the simultaneous-open-handles problem. Since the files are re-opened automatically if they're needed again, the only tradeoff is maybe a few milliseconds spent opening the file.

Definitions

DOS Handle

The number assigned by DOS for use in accessing disk files, on my system a number from five to 19.

F-PC Handle

The address that contains Zimmer's 70-byte file path and name, attribute, and DOS handle. I've added two bytes to each F-PC handle when defined with the `FILE:` word to store a link to the previous file.

Record Control Block (RCB)
An array of six bytes embedded in the code of each record definition that contains the address of the respective F-PC handle, the number of bytes in each record, and the offset into the file of the first record.

Parent File
The file that contains your data for a given record name or group of records. You can have either one defined record per parent file, or you may have a group of records share the same file but occupying different locations in that file.

Alternatively, you can have the records overlap, if you are so inclined. I do this in at least one case: the real records are two bytes, but I also set up 1024-byte records that occupy the same data area for when I want to erase it all quickly.

Figure One.

```
file: book.dat
file: price.dat
file: stock.dat
file: account.dat

\ initial       parent                record
\  offset  size file                  name

       0    48   book.dat record: title     \ name of the book

       0     2   book.dat record: price     \ how much $ ?
\ two bytes will work for this if none of your book prices
\ exceeds $655.35

      0    2   stock.dat record: stock    \ how many of each book?

       0    32   account.dat record: account    \ customer name

    file: stat.dat

      0    2   stat.dat record: pointers  \ misc. pointers, values
     16    8   stat.dat record: purchases \ cumulative purchases

\ The eight bytes will give me room to track a dollar amount,
\ total # of orders, and the last order date:

       4 derive >#orders
       6 derive >latest
```

Some values you may want to change:

B/BUF	Set this to the largest size record you're likely to use. If you need to enlarge it later, changing it won't affect any of your current files.
MAXFILES	The most files you're likely to ever use in your program. This is just used for error trapping.
#BUFS	The number of disk buffers you want. More buffers mean less frequent disk I/O, but also less security in case of power interruptions, DOS seizures, or coffee on the keyboard.

In order to set up your records, you will of course need to know what you intend to call each parent data file. The word `FILE:` will then create the structure for you. For example, executing
`FILE: NAMES.DAT`

will create the Forth word `names.dat` and code to support records using the `NAMES.DAT` file. You do not have to have this file on your disk at this point. It will be created (or truncated, if already present) when you execute `ESTABLISH`, i.e., `NAMES.DAT ESTABLISH`.

`FILE:` creates the F-PC handle, inserting any special path you might have stored in the `RPATH` string. This path is only necessary if you will be operating the program in a different directory from the data files.

The created file is chained to any previously created ones. The variable `CHAIN` contains the address of the most recently defined F-PC handle, and from there you can track down all of them by following the links until you reach zero.

In fact, the word `DO-CHAIN` is set up to do just that, executing the vectored `CHAIN` word for each file. It is used for closing files, changing paths, and displaying the chain.

The word `?OPEN` checks to see if a DOS handle is assigned. If the field is less than zero, it is assumed that the file is not open. When the file is closed, the DOS handle field is set to -1. Before a file is opened, a check is done to see if there is a drive and path name in the F-PC handle. If not, the current directory is inserted.

`FILE-ERR?` displays the name of the file if there is a problem opening or closing it.

When an error occurs while accessing a record (not opening or closing a file), ?RERR will display the DOS disk error code and any read/write error code, along with the record name (not the file) being accessed.

I'm not going to go into a detailed explanation of the F83 buffer system here, since it is very prevalent and I only understand it for a few minutes at a time anyway, but I have changed the file pointer in the buffer descriptor array (the area around `FIRST`; `RFIRST` in this code) so that it points to the RCB instead of the FCB of the relevant file. That way, when a read or write operation is called for, all the needed data (DOS handle, record length, etc.) is right there for processing.

When I got hold of F-PC, I thought that maybe Zimmer's block buffer code might work even better with some minor adaptations, but I already had this written and the assembly coded definitions intimidated me. Perhaps someone else can let me know how to make it fit?

Reading a Record From Disk

When it's time for a record to be read in, here is the sequence:

MISSING saves out the oldest record (if updated) and (RBUF) returns the buffer address. REC> reads the data in by accessing the RCB and calling REC-READ, which moves the file pointer (via DISKPREP) and attempts to read the record.

If REC-READ discovers that the proper number of bytes were not read, and no other error was encountered, (EXTEND?) is executed. (EXTEND?) checks to see how many bytes need to be appended to the file to include the requested record.

If the file needs to be enlarged, one of the following actions is taken:

- If MONITOR equals zero, the record is read in automatically.
- If MONITOR equals -1 (true), a dialog box appears asking you if you want to append the file the needed amount. If you select *no*, the program aborts.
- If MONITOR equals 1, the dialog box only appears if the amount to append exceeds the value you assigned EXTENT, which is the minimum increment used to extend files. This would normally happen only if you are not adding records sequentially (and/or your parameter stack gets mixed up.)

I recommend that you leave MONITOR on, at least until you get most of the bugs worked out of your code; it will save you some debugging time.

After the file is extended, the read operation is re-performed which places your record in one of the buffers. Note that records are always extended during read operations only; you've got to read the record into a buffer before you have an address to store data. You shouldn't get any errors during write operations (even when the disk is full), unless there is a disk problem.

Accessing the Records

To read in a record, simply place the record number on the stack and invoke the record name. Supposing you had defined a 14-byte record called PHONE#, the phrase

```
5 PHONE#              ( -- adr )
```

would read the fifth record into a buffer, returning the buffer address. If you modify the data at this address (and execute UPDATE), it will be saved out when its buffer is needed for something else.

The word THE is helpful at times. It will return the size as well as the buffer address of the record specified. For example,

```
7 THE PHONE#          ( -- adr ct )
```

returns the address and length of the seventh record. If this were stored in text format, you could then execute TYPE.

IT is a vectored word. Initialize it with the CFA of your record name.

```
' PHONE# IS IT
```

would set things up so that subsequent execution of IT would return the buffer address and byte count of your phone# record. In that case, these two phrases would produce the identical result:

```
3 the phone#        ( -- adr ct )
3 it                ( -- adr ct )
```

I use IT mostly for sorting routines; the same routine can sort a variety of records simply by patching IT.

SIZE> returns the size of the in-line record name without actually reading the disk. Usage would be:

```
size> phone#       ( -- ct )
```

The three words THE, IT, and SIZE> are all state smart. They will, therefore, work the same interactively as they do inside a definition.

DEL-PATHS is provided to allow you to delete any paths that might have been prepended to your F-PC handles. This is in case you want to change data directories. I keep records from previous years on floppies, and my current year on the hard drive. When I want to change years, I call this word and change the logged drive.

Debugging/Information Tools

The word BUFS allows you to see which records are currently in the buffers. Information listed is the record number, the record name, and the drive/path/file name. If the record has been updated, an asterisk is displayed after the record number. The records are listed in order of the most recently accessed—therefore, the last one you see will be overwritten when the next record is read from the disk.

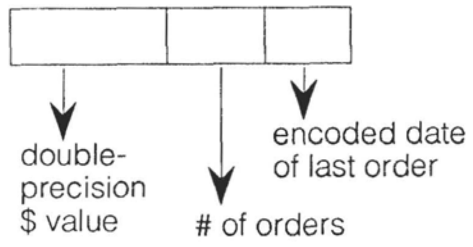
.CHAIN is provided to show all the defined parent files. The information displayed here is the DOS handle and the drive/path/file name. If no drive/path is present, it means that you have not accessed that record yet (or have executed DEL-PATHS). Once a record is opened, the path name remains—even after being closed.

An Example

Just to give you an idea of how this all works, suppose you were designing program for a mail-order bookstore. You might set up the files and records shown in Figure One.

At the risk of confusing things, I've combined three records into one record structure (PURCHASES) to illustrate how you can cut down on the number of data files, if you so desire. Grouping the three data structures together under one record name like this allows you to have an unlimited number of each. If you had defined them as distinct records starting at different offsets in the same file, you would run the risk of not allowing enough space for customer-base growth, not to mention having to create a large file before you have any data to enter.

Data structure:



The way to retrieve the number of orders filled for customer #15 is to execute:

```
15 purchases >#orders @
```

The alternative way of keeping these all in the same file would be to decide on a maximum number of customers (10000 for this example) and do the following:

```
    0  2   stat.dat record: pointers
             \ this doesn't change
    6  4   stat.dat record: purchases
             \ just the dollar amount now
40006  2   stat.dat record: #orders
60006  2   stat.dat record: latest
```

The trouble is, I've now limited myself to 10,000 customers and also am starting out with a 60K file before my first order! This problem doesn't arise, of course, if you don't combine records in a file.

If you combine many records, the word ALLOW will help you save some math. Using it, the above example becomes:

```
0  2      3 allow stat.dat record: pointers
   4  10000 allow stat.dat record: purchases
   2  10000 allow stat.dat record: #orders
   2             stat.dat record: latest
```

For keeping track of books and customers, let's create three records to work with:

```
labeling pointers  \ These are equivalent:
0 label: #accounts
  \ : #accounts  0 pointers ;
1 label: #titles
  \ : #titles    1 pointers ;
2 label: #books
  \ : #books     2 pointers ;
```

Now to enter some data.

Execute the following to create the necessary files on your disk:

```
   book.dat establish
  price.dat establish
  stock.dat establish
account.dat establish
   stat.dat establish

\ Zero out your customers and inventory:
#accounts off update
#titles   off update
#books    off update

\ You could define the following word to
\ add new books:
: titles! ( -- )
     begin  cr ." Title: "
            #titles @ the title
            2dup blank
            expect
            span @
     while  update
            #titles +dsk
     repeat ;
```

This allows you to enter book names until you press a <cr> without any entry. You can add the prices later.

To type the name of book #1, enter:

```
1 the title type <cr>
```

To see how many titles you have entered:

```
#titles ? <cr>
```

It couldn't get much simpler! You can exit F-PC (by typing BYE), restart it again, and your data will still be there.

CAUTIONS

The following situations have been carefully tested and found to exhibit undesirable behavior:

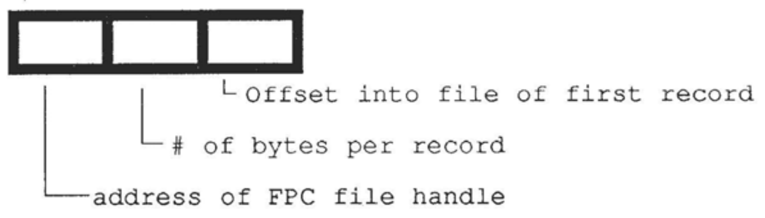
1. If any files have been appended, FLUSH must be executed before terminating F-PC. This is done automatically if you use BYE, but if DOS seizes up or you crash your program you'll most likely end up with lost clusters where your new records used to be.
2. If you load your program, FORGET part of it, then re-load, you may wind up with a problem when you execute FLUSH or .CHAIN, since FORGET doesn't tell CHAIN which links were forgotten. To avoid problems here, if you FORGET *any* parent files at all, forget them *all* and execute CHAIN OFF before re-loading. In fact, it's probably a good idea to put CHAIN OFF just before your first FILE: command.
3. If you do #2 (above) without FLUSHing between reloadings, you'll soon use up all your DOS handles if you access any records in between. This is because the original handles were forgotten before they were closed. You will suddenly be unable to open your files.

Note: This source code is available in the FIG software library on GEnie, file# 2586, RNDMRECS.ZIP.

```
comment:
```

Each named record has a Record Control Block (RCB) associated with it (not to be confused with an FCB) which contains the following info:

Record Control Block:



Each file structure consists of an FPC handle with 2 bytes appended to link to the previous file. This allows all files to be closed at once using FLUSH.

Zimmer's FPC handle structure:

```
The HANDLE memory data structure is as shown here.        2 bytes
  1byte     65 bytes        2 bytes     2 bytes            [ link to
[ count   ] [ name....0 ] [ attrib ] [ handle > -1 ]        previous
  addr        addr+1          addr+66     addr+68           handle in
               |               |           |                chain ]
               └─ NAM          └─ ATTRIB   └─ HNDLE           |
    └─ Address of the array returned by a word                |
       defined with HANDLE.                                   |
                                                    I've added this part
                                                    for the chained parent
                                                    files.
```

```
The address of this handle is returned by the word >RHANDLE given the address
of the particular Record Control Block.
comment;

read-write  def-rwmode
1024 constant b/buf    \ set this to your maximum expected record size
create rpath 64 allot    rpath  off

\ create record family                              91-04-25 brs

variable chain

60 constant maxfiles     \ this can be anything you like; a larger number
                         \ won't use up any more memory.  It's just a
                         \ precaution against the links being messed up.

: file:   ( -- )   \ usage =   FILE: filename.ext
         >in @
         handle here b/hcb -
         chain @ , dup chain !
         swap >in !
         dup bl word count rot place
         rpath prepend.apath drop ;

: derive   ( n -- )    \   run-time:  ( addr -- addr+n)
  create ,
  does> @ + ;
```

```
comment:
  Define a word to convert the address to the value offset by n.
  This is in case you want to access just a piece of a record.
comment;


b/hcb derive >flink   ( adr -- adr' )    \ convert to link address

defer chained  ( adr -- )

: do-chain ( -- )
  chain @ dup
  if maxfiles 0
     do dup chained
        >flink @ dup 0=
        ?leave
     loop
  then
  abort" Error in File chain" ;

: (.chain)   ( adr -- )
  cr dup >hndle @ dup -1 >
  if 6 .r
  else drop ."      --"
  then
  2 spaces
  count type ;

: .chain    ( -- )
  cr cr chain @
  if ." Handle  Filename"
      cr ." ================="
     ['] (.chain) is chained
     do-chain
  else  ." No Files Chained"
  then
  cr ;


\ access the F-PC handle from the soon-to-be-defined Record Control Block

' @ alias >rhandle ( rcb -- adr )          \ this points to the F-PC handle

: @rhandle   ( rcb -- n )                  \ return the DOS handle
  >rhandle >hndle @ ;

: .rname   ( rcb -- )
  body> >name .id ;                        \ type data name


\ Open and Close files

: file-err?  ( hcb f adr ct  -- ) rot      \ display name of the DOS FILE
  if cr beep                               \ if there is an error.
     type ." in "  count type
     abort
  else 3drop
  then ;

: ?open-err   ( hcb f -- )
  " Open error " file-err? ;

: ?close-err  ( hcb f -- )
  " Close error " file-err? ;

: ?open   ( rcb -- )                       \ ensure that the file is open
       dup @rhandle 0<                     \ is no handle there?
       if   >rhandle dup hopen  ?open-err  \ if so, open the file
       else drop                           \ otherwise, nevermind
       then ;
```

```
: rclose ( parentadr  -- )                    \ close this handle
         dup hclose ?close-err ;


: ?rerr  ( rcb f -- )
  disk-error @ or
  if   cr ." Disk/File error " disk-error ? rwerr ?
       .rname abort
  else drop
  then ;

comment:
  ?RERR will display the name of the RECORD in the event of an error
  during disk access.  This helps you to determine the offending calling
  process.  Compare to FILE-ERR?, above, which just tells you if a problem
  occurred during opening or closing.
comment;

: (roffset)   ( rec# rcb -- d )
         2+ 2@ rot um* rot 0 d+ ;

defer  roffset               \ deferred in case I want to convert back to my
                             \ old CP/M format for some reason.
' (roffset) is roffset

: locate ( rec# rcb -- )
  tuck roffset rot
  dup ?open
  >rhandle movepointer ;     \ aim at the record

: diskprep  ( dest rec# rcb -- rcb ct dest ct handle-adr )
  -rot pluck dup>r locate
  r@ 2+ @ tuck
  r> >rhandle ;              \ prepare all necessary data for disk i/o


\ read/write a record

defer extend?  ' 3drop is extend?

: rec-read  ( dest rec# rcb -- )
  rwerr off
  3dup diskprep hread  <>    \ point and shoot
  if rwerr @  ?rerr extend?
  else 3drop drop
  then ;

: rec-write  ( source rec# rcb -- )
  diskprep hwrite <>  ?rerr ;



\ automatic (or semi) record extension

b/buf 8*  constant extent   \ the minimum space taken by any file on my disk

: ?extend-err  ( hcb f -- )
  " Append error " file-err? ;

                                     \ d1 = new eof
: larger    ( rec# rcb -- d1 d2 f )  \ d2 = how much bigger the file should be
  dup>r                              \  f = true if d2 > extent
  dup 2+ @ >r  \ length
  roffset r> 0 d+
  2dup extent um/mod drop
  negate extent + 0
  d+
  2dup r> >rhandle endfile d-        \ calculate size increase needed
  2dup extent 0 d> ;                 \ is it bigger than EXTENT ?
```

```
: (extend) ( d hcb -- )                \ extend file by writing a zero at the
  dup>r -rot                           \ last two bytes of the desired file end
  2. d- r@ movepointer
  ['] false >body
  2 r> hwrite 2 <> ?extend-err ;

variable monitor  monitor on      \ monitor on = all file extensions verified
                                  \ monitor off = no permission needed
                                  \ monitor = 1  -- only get permission when
                                  \    you need to extend file > one EXTENT

: permission  ( d hcb -- )
  savecursor savescr
  >attrib4
  20 5 75 10 box&fill beep
  ." Is it okay to extend the file" bcr
  count type  space d. ." bytes? (Y/N) "
  key upc 'Y' <>  bcr
  >norm
  abort" Program aborted"
  restscr restcursor ;


: (extend?)  ( dest rec# rcb -- )
  dup >rhandle >r
  2dup larger 2 or
  monitor @  and
  if r@ permission
  else 2drop then
  r> (extend)  diskprep hread <> ?rerr ;

' (extend?) is extend?


\ Basic record access:
\ This stuff is all pretty much straight out of the F83 block buffer system.

4 constant #bufs

#bufs 1+ 8* 2+ constant >rsize

create >bufs ( -- adr )  >rsize allot    #bufs b/buf * allot

: >end  ( -- adr )
  >bufs >rsize 2- + ;

: buf# ( n -- adr )
  8* >bufs + ;

: >upd    ( -- adr )
  1 buf# 6 + ;

>bufs >rsize + constant rfirst

rfirst #bufs b/buf * + constant rlimit

: latest? ( n rcb -- rcb n f1 | a f f1 )
  disk-error off
  swap 2dup 1 buf# 2@ d=
  if  2drop 1 buf# 4 + @  false true
  else false
  then ;

: absent?  ( n rcb -- a f )
  latest? ?exit  false #bufs 1+ 2
  do drop 2dup i buf# 2@ d=
    if 2drop i leave
    else false
    then
    loop  ?dup
    if buf# dup >bufs 8 cmove
       >r >bufs  dup 8 + over r> swap  - cmove>
       1 buf# 4 + @ false
    else  >bufs 2!  true
    then ;

  : discard   ( -- )
    1  >upd ! ;

  : >prep   ( adr -- buffer rec# rcb ) \ use the buffer pointer to find the
    dup 4 + @
    swap  2@ swap ;                    \ necessary addresses/values

  : missing   ( -- )
    >end 2- @ 0<
    if  >end 2- off
        >end 8 - >prep  rec-write
    then
    >end 4 - @ >bufs 4 + !
    1 >bufs 6 + !
    >bufs  dup 8 + #bufs 8* cmove>  ;

  : (rbuf)   ( n rcb -- adr )
    absent?
    if missing 1 buf# 4 + @ then ;

  : rec>     ( n rcb -- adr )
    (rbuf)  >upd  @ 0>
    if  1 buf# dup >prep  rec-read
        6 + off
    then ;

  : record:    ( offset, b/rec file-- ) \ <name> ¦ run =  ( n -- adr )
    create , , ,
      does>  rec> ;

  comment:
    When the record is created, the following fields are laid down:

    1.  The parental file hcb, i.e. which file will store the data
    2.  How many bytes to allot for each record (since it's random access)
    3.  How far from the beginning of the file (in bytes) will these records
        start?

    The Record Control Blocks & buffers keep track of what's going on; all you
    need do is call the name, read/write data to the address, and UPDATE as
    needed.  Just like the F83 screen buffers, each updated record is
    automatically saved to disk when it's buffer is needed.
  comment;


  : allow   ( offset rlen #recs -- offset' offset rlen )
    >r 2dup r> * + -rot ;

  comment:
    ALLOW just makes things a little easier when you're defining records.
    It's purpose is to calculate the number of bytes to "allow" before
    starting the next record in the file.

    It leaves the next offset on the stack, ready to use for the next record.
  comment;


  : empty-buffers   ( -- )
    rfirst  rlimit over - erase
    >bufs #bufs 1+ 8* erase
    rfirst 1 buf#
    #bufs 0
    do dup on
```

```
      4 + 2dup !
      swap b/buf +
      swap 4 +
   loop
   2drop ;

empty-buffers

: save-buffers  ( -- )
   1 buf#  #bufs 0
   do  dup @ 1+
     if  dup 6 + @ 0<
       if dup >prep  rec-write
          dup 6 + off
       then
       8 +
     then
   loop drop ;

: flush  ( -- )
   save-buffers
   empty-buffers
   ['] rclose is chained
   do-chain ;

   ' flush is byefunc

: update ( -- )
   >upd on ;

: update:  ( -- )    \ runtime: ( ? -- )
   create ' ,
   does> perform  update ;

update: dsk!  !
update: dsk2! 2!
update: dskc! c!
update: +dsk  incr

comment:
   the DSK words are just versions of !, 2!, c!, etc. which mark the record
   as being updated.
comment;

: (size)    ( -- ct )
   >body 2+ @ ;

: (it)   ( n cfa -- adr ct )  \ address & count of cfa's nth record
   dup (size)                 \ (cfa=word defined with RECORD:)
   >r execute r> ;

: it   ( n -- adr ct )        \ return the address & ct of the Nth record
   does>  @ (it) ;  it        \ initialize with:  ' MYREC IS IT

: the ( NAME ¦ n -- adr ct ) \ return adr/ct of nth record
   state @
   if [compile] [']
      compile (it)
   else ' (it)  then ; immediate

: size> ( NAME ¦ -- ct )     \ return the record size without executing it
   state @
   if [compile] [']
      compile (size)
   else ' (size) then ;   immediate

: label: ( n -- )  \ run: ( -- adr )  \ define a named record @ record# n
   create @> it , ,
   does> 2@ execute ;
```

Advertisers Index

The Computer Journal	*25*
FORML Conference	*34*
Forth Interest Group	*40, 44*
Harvard Softworks	*27*
Laboratory Microsystems	*20*
Miller Microcomputer Services	*25*
Silicon Composers	*2*

```
: labeling   ( NAME  | -- )   \ the record to LABEL:   ex: LABELING MYREC
  ?exec
  ' ['] it >is ! ;


: establish   ( adr -- )   hcreate abort" File creation problem" ;

comment:
 Truncate (or create) a file.  It's length is set to zero.
 use this to initialize (create) a new file on the disk.
 EXAMPLE:  mystuff.dat establish
comment;

: path-len ( adr ct -- n )
  tuck 1- over +
  do i c@ '\' =
     ?leave
     1-
  -1 +loop ;

: -path   ( handle -- )
  dup count
  2dup path-len
  ?dup
  if 2dup - >r
     /string
     2 pick place
     count + r> erase
  else 3drop
  then ;

: del-paths ( -- )
  flush
  ['] -path is chained
  do-chain ;
```

```
\ delete all paths from the chained F-PC handles


: bufs   ( -- ) \ Display which records are in the buffers
  1 buf# #bufs 1+ 1
   DO CR I . 2 SPACES
      DUP @ TRUE =          \ FFFF=empty buffer
      OVER 6 + @ 1 = OR     \ waiting to be reread
      IF ."  ---"
      ELSE DUP @ 4 .R   SPACE
           dup 6 + @
           if ascii *       \ if updated
           else bl then
           emit space
           dup 2+  @ dup  .rname
           30 #out @ - spaces
           >rhandle count type
      THEN 8 +
   LOOP DROP CR ;
```

Total control with *LMI FORTH*™

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

For Development:
Interactive Forth-83 Interpreter/Compilers for MS-DOS, OS/2, and the 80386
- 16-bit and 32-bit implementations
- Full screen editor and assembler
- Uses standard operating system files
- 500 page manual written in plain English
- Support for graphics,floating point, native code generation

For Applications: Forth-83 Metacompiler
- Unique table-driven multi-pass Forth compiler
- Compiles compact ROMable or disk-based applications
- Excellent error handling
- Produces headerless code, compiles from intermediate states, and performs conditional compilation
- Cross-compiles to 8080, Z-80, 8088, 68000, 6502, 8051, 8096, 1802, 6303, 6809, 68HC11,34010, V25, RTX-2000
- No license fee or royalty for compiled applications

Laboratory Microsystems Incorporated
Post Office Box 10430, Marina del Rey, CA 90295
Phone Credit Card Orders to: (213) 306-7412
FAX: (213) 301-0761

Structured Pattern Matching

Ariel Scolnicov
Mevasseret Zion, Israel

Forth is not generally regarded as a suitable language for string processing. This is rather strange: Forth is, as we keep telling ourselves, an *extensible* language. The problem of string handling has been addressed elsewhere. However, handling strings is not the whole issue. True string-processing languages, such as AWK and SNOBOL, not only handle strings but also provide functions for string matching. These allow us to match a string against a *pattern*, a description of an entire class of strings. I have developed a set of words allowing the definition of patterns, along with the necessary routines to match patterns to strings.

At first, I considered using regular expressions for my patterns. Their main advantage is that they can be compiled as a finite-state machine, which can then be executed very quickly. Unfortunately, regular expressions (and finite-state machines) are extremely limited: while general enough to describe, say, a floating-point number (as [+-]?[0-9]+(\.[0-9]+)?([Ee][0-9]+)?), they are hardly intelligible. Worse, regular expressions can't "count": for instance, no regular expression exists which will match only strings with balanced braces. AWK, which uses regular expressions for its patterns, is limited in this respect.[1]

SNOBOL provides a far better pattern language. Unfortunately, the rest of SNOBOL is totally unstructured, with a goto as the only control structure, and a funny way of defining functions. What I really wanted was the string-pattern-matching features of SNOBOL. What I ended up writing has all the functionality of SNOBOL's pattern matcher, with a uniquely Forth-ish syntax.

Patterns

The pattern matcher is divided between two files: LOGIC.SEQ provides a very general "search driver," and STRMATCH.SEQ provides words to describe pieces of the string. Building a pattern is very Forth-like: patterns are built into the dictionary space, and every pattern is described by its address. Primitive patterns are created without arguments; logical connectors connect patterns found on the stack. At the end of a definition, a single address left on the stack represents the entire pattern. This address may be stored in a constant.

1. But to do AWK justice, it is just one of the impressive array of text-handling tools in the C/UNIX world. There, string parsing is supported by LEX and YACC.

Primitive patterns

`m" ..."` `( -- pat )`
Builds a pattern which matches the specified string.

`anyof" ..."` `( -- pat )`
Builds a pattern which matches if the current character is in the specified string.

`m' ...', anyof' ... '`
Like the previous two, but can contain the double quote character.

`POS, RPOS` `( n -- pat )`
Build patterns which succeed only if the current position is n characters from the start (end) of the string.

`HEAD, TAIL`
Patterns which match the beginning (end) of the string.

Operators

`&&` `( pat1 pat2 -- pat3 )`
Builds a pattern which matches pat1 followed by pat2 (concatenation).

`||` `( pat1 pat2 -- pat3 )`
Builds a pattern which tries to match pat1. If the match or any succeeding match fails, goes back (backtracks) and tries to match pat2 (alternation).

`[&`
Starts a concatenation chain.

`&]`
Closes a concatenation chain. All patterns between `[&` and `&]` are linked with `&&`. `[& a b c &]` is equivalent to `a b c && &&`.

`[|`
Starts an alternation chain.

`|]`
Closes an alternation chain. All patterns between `[|` and `|]` are linked with `||`.

```
OPT         ( pat -- pat? )
```
Builds a pattern which optionally matches pat.

```
~~          ( pat -- !pat )
```
Builds a pattern which matches anything but pat. If successful, matches the shortest substring which cannot be continued to match pat (negation).

```
MANY        ( pat1 -- pat2 )
```
Matches several (possibly 0) copies of pat1. The number of copies matched is the least needed to enable the rest of the pattern to match. Since MANY matches the *least* number of copies, a pattern should be concatenated to it to prevent MANY matching the null string.

```
MOST        ( pat1 -- pat2 )
```
Like MANY, but matches the maximum number of copies possible. This can sometimes take a lot of time, so MANY is generally preferable.

A Few Examples

```
m" abc" m" def" &&
```
matches "abcdef"

```
m" abc" m" def" ||
```
matches "abc" or "def"

```
[& m" Th"
[| m" eir" m" at" m" ere" |] m" !" &]
```
matches "Their!", "That!" or "There!"

```
anyof" 0123456789" many
```
matches "123", "0123", etc., but if unanchored will match "" (the null string)

```
[& m" Good" m" bye" ~~ &]
```
matches "Goodl" from "Goodly" or "Good " from "Good riddance", but fails to match "Goodbye"

```
[& m" xy" many m" xyz" &]
```
matches "xyz", "xyxyz", "xyxyxyz", etc.

```
[& m' xy' most m' xyz' &]
```
as above

Note the behaviour of MANY in the last example: m" xy" most seems to avoid matching the final "xy", so as to enable m" xyz" to successfully match. This seemingly intelligent behaviour is characteristic of all the pattern-matching functions which involve a choice (||, MANY, MOST). What actually happens is a search between all the different possibilities. This is explained in further detail in the next section.

Three special patterns are defined, which are sometimes useful:

FAIL	Fails to match, causes a backtrack.
NULL	Matches the null string, always succeeds.
CUT	Prevents backtracking behind its use.

CUT "freezes" the current state of matching, and doesn't let subsequent backtracking change it. It is a powerful tool for speeding up long matches, but should be used with extreme caution.

Structuring Patterns

As mentioned before, patterns can be stored as constants. This allows us to structure our patterns. For example, to match a series of numbers, use:

```
anyof" 0123456789" constant digit
digit digit most && constant number
number [& m" " m"  " many number &] most
constant number-list
```

Here, a number is defined as at least one digit, and a number list as at least one number, with successive numbers separated by at least one space. This structure makes patterns more readable. Instead of the unreadable regular expression for a floating-point number, we write:

```
[& number  m" ." number && opt
anyof" Ee" number && opt &]
constant fp-number
```

The pattern language described up to here is very useful, but it isn't really more powerful than regular expressions. In order to be able to count, we need recursion: the ability to call the current pattern, or to forward reference a pattern.

In Forth, this is usually done by using DEFERed words or variables containing the execution address. At compile time, code to fetch the value is laid down; at execution time, the actual address is fetched. The pattern matcher contains @CALL, which works like PERFORM:

```
@CALL      ( v -- pat)
```
Returns a pattern which will call the pattern stored in variable v. Always succeeds.

So we can write (using some of the previous definitions):

```
variable list
number list @call || constant item
item [& m"  " m"  " many item &]
many constant items
m" (" items m" )" list !
```

This matches lists, where a list may contain numbers or further lists, but no null list is permitted.

The remaining three words don't perform any pattern-matching actions, but instead allow the user's words to be executed during the matching operation. Obviously, these words cannot leave values on the stack, since the pattern matcher uses the stack.

Instead, an *auxiliary stack* is provided, accessed by >AUX and AUX>. The auxiliary stack is restored when backtracking,

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.
Most items list three different price categories: USA, Canada, and Mexico / Other countries via surface mail / Other countries via air mail
Note: Where only two prices are listed, surface mail is not available.

"Were Sure You Wanted To Know..."

***Forth Dimensions*, Article Reference** 151 - $4/5
★ An index of Forth articles, by keyword, from *Forth Dimensions* Volumes 1–13 (1978–92).

FORML, Article Reference 152 - $4/5
★ An index of Forth articles by keyword, author, and date from the FORML Conference Proceedings (1980–90).

FORTH DIMENSIONS BACK VOLUMES

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

Volume 1 Forth Dimensions (1979–80) 101 - $15/16/18
Introduction to FIG, threaded code, TO variables. fig-Forth.

Volume 2 Forth Dimensions (1980–81) 102 - $15/16/18
Recursion, file naming, Towers of Hanoi, CASE contest, input number wordset, 2nd FORML report, FORGET, VIEW.

Volume 3 Forth Dimensions (1981–82) 103 - $15/16/18
Forth-79 Standard, Stacks, HEX, database, music, memory management, high-level interrupts, string stack, BASIC compiler, recursion, 8080 assembler.

Volume 4 Forth Dimensions (1982–83) 104 - $15/16/18
Fixed-point trig., fixed-point square root, fractional arithmetic, CORDIC algorithm, interrupts, stepper-motor control, source-screen documentation tools, recursion, recursive decompiler, file systems, quick text formatter, ROMmable Forth, indexer, Forth-83 Standard, teaching Forth, algebraic expression evaluator.

Volume 5 Forth Dimensions (1983–84) 105 - $15/16/18
Computer graphics, 3D animation, double-precision math words, overlays, recursive sort, a simple multi-tasker, metacompilation, voice output, number utility, menu-driven software, vocabulary tutorial, vectorerd execution, data acquisition, fixed-point logarithms, Quicksort, fixed-point square root.

Volume 6 Forth Dimensions (1984–85) 106 - $15/16/18
Interactive editors, anonymous variables, list handling, integer solutions, control structures, debugging techniques, recursion, semiphores, simple I/O words, Quicksort, high-level packet communications, China FORML.

Volume 7 Forth Dimensions (1985–86) 107 - $20/22/25
Generic sort, Forth spreadsheet, control structures, psuedo-interrupts, number editing, Atari Forth, pretty printing, code modules, universal stack word, polynomial evaluation, F83 strings.

Volume 8 Forth Dimensions (1986–87) 108 - $20/22/25
Interrupt-driven serial input, data-base functions, TI 99/A, 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 - $20/22/25
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 - $20/22/25
dBase file access, string handling, local variables, data structures, object-oriented Forth, linear automata, standalone applications, 8250 drivers, serial data compression.

Volume 11 Forth Dimensions (1989–90) 111 - $20/22/25
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 - $20/22/25
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.

1980 FORML PROCEEDINGS 310 - $30/31/40
Address binding, dynamic memory allocation, local variables, concurrency, binary absolute & relocatable loader, LISP, how to manage Forth projects, n-level file system, documenting Forth, Forth structures, Forth strings.

1981 FORML PROCEEDINGS 311 - $45/48/55
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.

1982 FORML PROCEEDINGS 312 - $30/31/40
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.

1983 FORML PROCEEDINGS 313 - $30/32/40
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.

1984 FORML PROCEEDINGS 314 - $30/33/40
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.

1985 FORML PROCEEDINGS 315 - $30/32/40
Threaded binary trees, natural language parsing, small learning expert system, LISP, LOGO in Forth, Prolog interpreter, BNF parser in Forth, formal rules for phrasing, Forth coding conventions, fast high-level floating point, Forth component library, Forth & artificial intelligence, electrical network analysis, event-driven multitasking.

1986 FORML PROCEEDINGS 316 - $30/32/40
Threading techniques, Prolog, VLSI Forth microprocessor, natural-language interface, expert system shell, inference engine, multiple-inheritance system, automatic programming environment.

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

1987 FORML PROCEEDINGS 317 - $40/43/50
Includes papers from '87 euroFORML Conference. 32-bit Forth, neural networks, control structures, AI, optimizing compilers, hypertext, field and record structures, CAD command language, object-oriented lists, trainable neural nets, expert systems.

1988 FORML PROCEEDINGS 318 - $24/25/34
Human interfaces, simple robotics kernel, MODUL Forth, language topics, hardware, Wil's workings & Ting's philosophy, Forth hardware applications, ANS Forth session, future of Forth in AI applications.

1988 AUSTRALIAN PROCEEDINGS 380 - $24/25/34
Proceedings from the first Australian Forth Symposium, held May 1988 at the University of Technology in Sydney. Subjects include training, parallel processing, programmable controllers, Prolog, simulations, and applications.

1989 FORML PROCEEDINGS 319 - $40/43/50
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.

1990 FORML PROCEEDINGS 320 - $40/43/50
Forth in industry, communications monitor, 6805 development. 3-key keyboard, documentation techniques, object-oriented programming, simplest Forth decompiler, error recovery, stack operations, process control event management, control structure analysis, systems design course, group theory using Forth.

BOOKS ABOUT FORTH

ALL ABOUT FORTH, 3rd ed., June 1990, Glen B. Haydon 201 - $90/92/105
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.

THE COMPLETE FORTH, Alan Winfield 210 - $14/15/19
A comprehensive introduction, including problems with answers (Forth-79).

eFORTH IMPLEMENTATION GUIDE, C.H. Ting 215 - $25/26/35
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. (w/disk)

F83 SOURCE, Henry Laxen & Michael Perry 217 - $20/21/30
A complete listing of F83, including source and shadow screens. Includes introduction on getting started.

FORTH: A TEXT AND REFERENCE 219 - $31/32/41
Mahlon G. Kelly & Nicholas Spies
A textbook approach to Forth, with comprehensive references to MMS-FORTH and the '79 and '83 Forth standards.

THE FORTH COURSE, Richard E. Haskell 225 - $25/26/35
This set of 11 lessons, called the Forth Course, is designed to make it easy for you to learn Forth. The material was developed over several years of teaching Forth as part of a senior/graduate course in design of embedded software computer systems at Oakland University in Rochester, Michigan. (w/disk)

FORTH ENCYCLOPEDIA, Mitch Derick & Linda Baker 220 - $30/32/40
A detailed look at each fig-Forth instruction.

FORTH NOTEBOOK, Dr. C.H. Ting 232 - $25/26/35
Good examples and applications. Great learning aid. poly-FORTH is the dialect used. Some conversion advice is included. Code is well documented.

FORTH NOTEBOOK II, Dr. C.H. Ting 232a - $25/26/35
Collection of research papers on various topics, such as image processing, parallel processing, and miscellaneous applications.

F-PC USERS MANUAL (2nd ed., V3.5) 350 - $20/21/27
Users manual to the public-domain Forth system optimized for IBM PC/XT/AT computers. A fat, fast system with many tools.

F-PC TECHNICAL REFERENCE MANUAL 351 - $30/32/40
A must if you need to know the inner workings of F-PC.

INSIDE F-83, Dr. C.H. Ting 235 - $25/26/35
Invaluable for those using F-83.

LIBRARY OF FORTH ROUTINES AND UTILITIES, James D. Terry 237 - $23/25/35
Comprehensive collection of professional quality computer code for Forth; offers routines that can be put to use in almost any Forth application, including expert systems and natural-language interfaces.

OBJECT ORIENTED FORTH, Dick Pountain 242 - $28/29/34
Implementation of data structures. First book to make object-oriented programming available to users of even very small home computers.

SEEING FORTH, Jack Woehr 243 - $25/26/35
"... 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 ..." **NEW**

SCIENTIFIC FORTH, Julian V. Noble 250 - $50/52/60 **NEW**
Scientific Forth extends the Forth kernel in the direction of scientific problem solving. It illustrates advanced Forth programming techniques with non-trivial 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. (Includes disk with programs and several utilities)

STACK COMPUTERS, THE NEW WAVE 244 - $62/65/72
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 (hardcover only).

STARTING FORTH (2nd ed.), Leo Brodie 245 - $29/30/38
In this edition of *Starting Forth*—the most popular and complete introduction to Forth—syntax has been expanded to include the Forth-83 Standard.

WRITE YOUR OWN PROGRAMMING LANGUAGE USING C++, Norman Smith 270 - $15/16/18
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.

ACM - SIGFORTH

The ACM SIGForth Newsletter is published quarterly by the Association of Computing Machinery, Inc. SIGForth's focus is on the development and refinement of concepts, methods, and techniques needed by Forth professionals.

Volume 1 Spring 1989, Summer 1989, #3, #4 910 - $24/26/34
F-PC, glossary utility, Euroforth, SIGForth '89 Workshop summary (real-time software engineering), Intel 80x8x. Metacompiler in cmForth, Forth exception handler, string case statement for UF/Forth. 1802 simulator, tutorial on multiple threaded vocabularies. Stack frames, duals: an alternative to variables, PocketForth.

Volume 2 #1, #2, #3, #4 920 - $24/26/34
ACM SIGForth Industry Survey, abstracts 1990 Rochester conf., RTX-2000. BNF Parser, abstracts 1990 Rochester conf., F-PC Teach. Tethered Forth model, abstracts 1990 SIGForth conf. Target-meta-cross-: an engineer's viewpoint, single-instruction computer.

Volume 3, #1 Summer '91 908 - $6/7/9
Co-routines and recursion for tree balancing, convenient number handling.

Volume 3, #2 Fall '91 909 - $6/7/9
Postscript Issue, What is Postscript?, Forth in Postscript, Review: PS-Tutor.

1989 SIGForth Workshop Proceedings 931 - $20/21/26
Software engineering, multitasking, interrupt-driven systems, object-oriented Forth, error recovery and control, virtual memory support, signal processing.

1990-91 SIGForth Workshop Proceedings 932 - $20/21/26
Teaching computer algebra, stack-based hardware, reconfigurable processors, real-time operating systems, embedded control, marketing Forth, development systems, in-flight monitoring, multi-processors, neural nets, security control, user interface, algorithms.

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.*

Prices: Each item below comes on one or more disks, indicated in parentheses after the item number. The price of your order is $6/9 per disk, or $25/37 for any five disks.

FLOAT4th.BLK V1.4 Robert L. Smith — C001 - (1)
Software floating-point for fig-, poly-, 79-Std., 83-Std. Forths. IEEE short 32-bit, four standard functions, square root and log. **IBM.**

Games in Forth — C002 - (1)
Misc. games, Go, TETRA, Life... Source. **IBM**

A Forth Spreadsheet V2, Craig Lindley — C003 - (1)
This model spreadsheet first appeared in *Forth Dimensions* VII, 1-2. Those issues contain docs & source. **IBM**

Automatic Structure Charts V3, Kim Harris — C004 - (1)
Tools for analysis of large Forth programs, first presented at FORML conference. Full source; docs incl. in 1985 FORML Proceedings. **IBM**

A Simple Inference Engine V4, Martin Tracy — C005 - (1)
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**

The Math Box V6, Nathaniel Grossman — C006 - (1)
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**

AstroForth & AstroOKO Demos, I.R. Agumirsian — C007 - (1)
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**

Forth List Handler V1, Martin Tracy — C008 - (1)
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**

8051 Embedded Forth, William Payne — C050 - (4)
8051 ROMmable Forth operating system. 8086-to-8051 target compiler. Incl. source. Docs are in the book *Embedded Controller Forth for the 8051 Family*. **IBM**

F83 V2.01, Mike Perry & Henry Laxen — C100 - (1)
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.**

F-PC V3.53, Tom Zimmer — C200 - (5)
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. Req. hard disk. **IBM, 83.**

F-PC TEACH V3.5, Lessons 0–7 Jack Brown — C201a - (2)
Forth classroom on disk. First seven lessons on learning Forth, from Jack Brown of B.C. Institute of Technology. **IBM, F-PC.**

VP-Planner Float for F-PC, V1.01 Jack Brown — C202 - (1)
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.**

F-PC Graphics V4.2f, Mark Smiley — C203a - (3)
The latest versions of new graphics routines, including CGA, EGA, and VGA suppport, with numerous improvements over earlier versions created or supported by Mark Smiley. **IBM, F-PC.**

PocketForth V1.4, Chris Heilman — C300 - (1)
Smallest complete Forth for the Mac. Access to all Mac functions, files, graphics, floating point, macros, create standalone applications and DAs. Based on fig & *Starting Forth*. Incl. source and manual. **MAC**

Yerkes Forth V3.6 — C350 - (2)
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**, System 7.01 Compatable.

JLISP V1.0, Nick Didkovsky — C401 - (1)
LISP interpreter invoked from Amiga JForth. The nucleus of the interpreter is the result of Martin Tracy's work. Extended to allow the LISP interpreter to link to and execute JForth words. It can communicate with JForth's ODE (Object-Development Environment). **AMIGA,** 83.

Pygmy V1.3, Frank Sergeant — C500 - (1)
A lean, fast Forth with full source code. Incl. full-screen editor, assembler and metacompiler. Up to 15 files open at a time. **IBM.**

KForth, Guy Kelly — C600 - (3)
A full Forth system with windows, mouse, drawing and modem packages. Incl. source & docs. **IBM, 83.**

ForST, John Redmond — C700 - (1)
Forth for the Atari ST. Incl. source & docs. **Atari ST.**

Mops V2.0, Michael Hore — C710 - (1)
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, docs & source. **MAC**

BBL & Abundance, Roedy Green — C800 - (4)
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. Req. hard disk. Incl. source & docs. **IBM HD hard disk reequired**

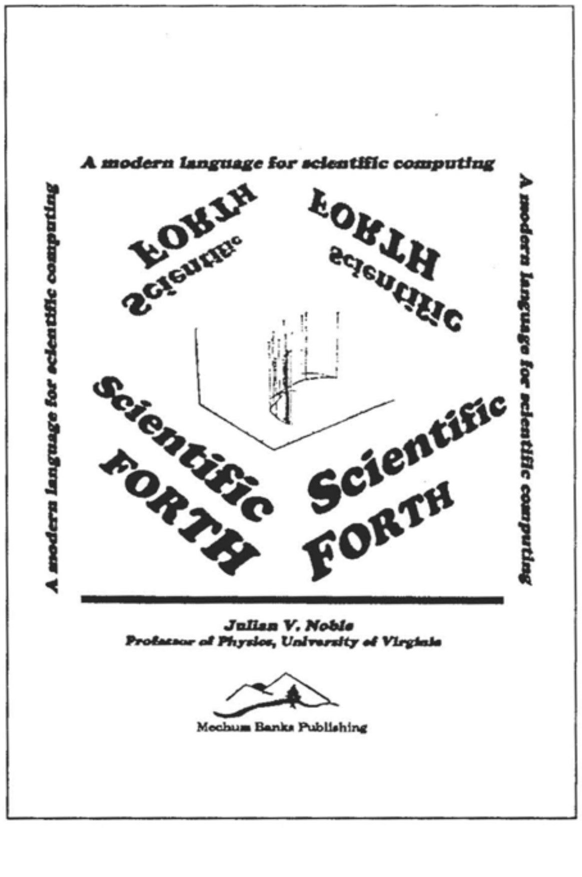

A modern language for scientific computing
Scientific FORTH
Scientific FORTH
Julian V. Noble
Professor of Physics, University of Virginia
Mechum Banks Publishing

fig-FORTH ASSEMBLY LANGUAGE SOURCE

Listings of fig-Forth for specific CPUs and machines with compiler security and variable-length names (see *Installation Manual*, below): - $15/16/18

1802	513 - March 81	**9900**	519 - March 81
6502	514 - September 80	**Apple II**	521 - August 81
6800	515 - May 79	**IBM-PC**	523 - March 84
6809	516 - June 80	**PDP-11**	526 - January 80
8080	517 - September 79	**VAX**	527 - October 82
8086/88	518 - March 81	**Z80**	528 - September 82

fig-FORTH INSTALLATION MANUAL 501 - $15/16/18
Glossary model editor—we recommend you purchase this manual when purchasing any of the source code listings above.

SYSTEMS GUIDE TO fig-FORTH 308 - $25/28/30
C. H. Ting (2nd ed., 1989)
How's and why's of the fig-Forth Model by Bill Ragsdale, internal structure of fig-Forth system.

MISCELLANEOUS

T-SHIRT "May the Forth Be With You" 601 - $12/13/15
(Specify size: Small, Medium, Large, Extra-Large on order form)
White design on a dark blue shirt.

POSTER (*Oct., 1980 BYTE* cover) 602 - $5/6/7

FORTH-83 HANDY REFERENCE CARD 683 - free

FORTH-83 STANDARD 305 - $15/16/18
Authoritative description of Forth-83 Standard. For reference, not instruction.

BIBLIOGRAPHY OF FORTH REFERENCES 340 - $18/19/25
(3rd ed., January 1987)
Over 1900 references to Forth articles throughout computer literature.

MORE ON FORTH ENGINES

Volume 10 January 1989 810 - $15/16/18
RTX reprints from 1988 Rochester Forth Conference, object-oriented cmForth, lesser Forth engines.

Volume 11 July 1989 811 - $15/16/18
RTX supplement to *Footsteps in an Empty Valley*, SC32, 32-bit Forth engine, RTX interrupts utility.

Volume 12 April 1990 812 - $15/16/18
ShBoom Chip architecture and instructions, Neural Computing Module NCM3232, pigForth, binary radix sort on 80286, 68010, and RTX2000.

Volume 13 October 1990 813 - $15/16/18
PALs of the RTX2000 Mini-BEE, EBForth, AZForth, RTX-2101, 8086 eForth, 8051 eForth.

Volume 14 814 - $15/16/18
RTX Pocket-Scope, eForth for muP20, ShBoom, eForth for CP/M & Z80, XMODEM for eForth.

Volume 15 815 - $15/16/18
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.

DR. DOBB'S JOURNAL

Annual Forth issue, includes code for various Forth applications.

Sept. 1982	422 - $5/6/7
Sept. 1983	423 - $5/6/7
Sept. 1984	424 - $5/6/7

FORTH INTEREST GROUP

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

Name ______________________
Company ______________________
Street ______________________
City ______________________
State/Prov. ____________ Zip ____________
Country ____________ Daytime phone ____________

OFFICE USE ONLY
By______ Date__________ Type________
Shipped by__________ Date__________
UPS USPS XRDS
Wt.__________ Amt.__________
BO By__________ Date__________
Wt.__________ Amt.__________

Item #	Title	Qty.	Unit Price	Total

☐ CHECK ENCLOSED (Payable to: Forth Interest Group)
☐ VISA ☐ MasterCard
Card Number ____________ Expiration Date ______
Signature ______________________

*MEMBERSHIP →

Sub-Total	
10% Member Discount, Member # ______	( )
**Sales Tax (CA only)	
Mail Order Handling Fee	$3.00
*Membership in the Forth Interest Group ☐ New ☐ Renewal $40/46/52	
* Enclosed is $40/46/52 for 1 full year's dues. This includes $36/42/48 for *Forth Dimensions.*	

MEMBERSHIP IN THE FORTH INTEREST GROUP

The Forth Interest Group (FIG) is a world-wide, non-profit, member-supported organization with over 1,500 members and 40 chapters. Your membership includes a subscription to the bi-monthly magazine *Forth Dimensions.* FIG also offers its members an on-line data base, a large selection of Forth literature and other services. Cost is $40 per year for U.S.A. & Canada surface; $46 Canada air mail; all other countries $52 per year. 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 from 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.

POSTAGE & HANDLING
Prices include shipping. The $3.00 handling fee is required with all orders.

SHIPPING TIME
Books in stock are shipped within seven days of receipt of the order. Please allow 4–6 weeks for out-of-stock books (deliveries in most cases will be much sooner).

**** CALIFORNIA SALES TAX BY COUNTY**
7.5%: Sonoma; **7.75%**: Fresno, Imperial, Inyo, Madera, Monterey, Orange, Riverside, Sacramento, San Benito, Santa Barbara, San Bernardino, San Diego, and San Joaquin; **8.25%**: Alameda, Contra Costa, Los Angeles San Mateo, Santa Clara, and Santa Cruz; **8.5%**: San Francisco; **7.25%**: other counties.

allowing it to contain state information.

`EXEC ( cfa -- pat )`
Returns a pattern which executes the given routine and matches the null string. The EXECed routine should have no stack action.

`PUSH`
A pattern which pushes the current position in the string onto the auxiliary stack, and matches the null string.

`SUBSTRING ( pat cfa -- pat )`
Returns a pattern which performs cfa if pat matched. The stack action of the routine should be `( adr len -- )`, where adr and len are the address and length of the substring which matched the pattern.

`AUX> ( -- n )`
Pops a value off the auxiliary stack.

`>AUX ( n -- )`
Pushes a value onto the auxiliary stack.

To load the pattern matcher, type:

```
fload logic  fload strmatch
```

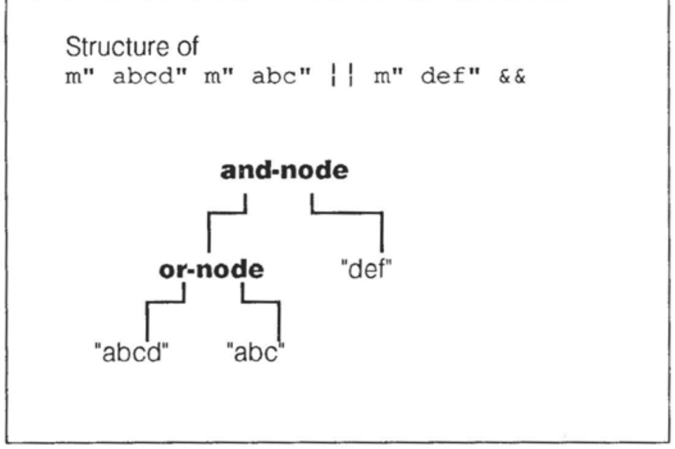
Two words are provided to actually perform the match:

`PMATCH ( adr len pat -- ?len flag )`
Attempts to match the string adr, len to pat. If the match succeeds, returns the length of the match and true. If it fails, returns false.

`PSEARCH`
`( adr len pat -- false or adr len true )`
Attempts to match pat to a substring of adr, len. If a match is found, returns the address and length of the match and true. If no match is found, returns false.

The pattern doesn't have to match the entire string. If you want to force the entire string to be matched, use `TAIL` at the end of the pattern.

Figure One.



As an example of the power of string pattern matching, EVAL.SEQ contains an expression evaluator. It is practically a direct translation of the grammar defining arithmetic expressions. Type `fload eval`, and then `eval 2+3*-(6-1)`.

Design

The design stage for this project was unusually long. I wrote at least two incorrect versions of the pattern matcher before I really understood pattern matching. Essentially, a pattern can be described as a tree: leaf nodes attempt to match characters in the string, or to perform some other primitive operation; other nodes concatenate or alternate patterns. Compilation of a pattern into a tree is straightforward; the problem is performing the pattern match.

Executing leaf nodes is simple: check if the characters at the current position match the leaf nodes. Concatenation nodes involve matching first one sub-pattern and, if it succeeds, the next one.

The tricky bit is alternation: the first sub-pattern should be executed; if the match or any subsequent match fails, the second sub-pattern should be tried. What is needed is a way of storing the *current state* of the matching process, so that we'll be able to return to it. Part of the answer is to push the state onto a special backtrack stack. Whenever a match fails, pop a state off that stack and continue execution from there.

What constitutes the "current state"? Consider matching the pattern `m" abcd" m" abc" || m" def" &&`

to the string "abcdef" (see Figure One for the pattern's structure). The first four characters are matched, but then `m" def"` fails. Backtracking, we match the first three characters. We now need to match the second half of the root `&&` node. Obviously, the current position in the string is part of the state. But the matcher also needs to know where to continue after the substring is matched.

The stumbling block here is that the current state has a variable size. Descending the pattern tree is a recursive process; this means we have to store return information on a call stack. The current state is the cursor position, along with the entire call stack.

The matching algorithm follows easily:

```
Empty call and backtrack stacks.
Push pattern's root onto call stack.
While call stack isn't empty:
    Pop node off call stack.
    Case of node:
        AND node:
            push 2nd and 1st subtrees onto call stack,
            succeed.
        OR node:
            push 2nd subtree onto call stack,
            push current state onto backtrack stack,
            replace 2nd subtree on call stack with 1st subtree,
            succeed.
        leaf node:
            match to string,
            succeed or fail accordingly.
    If failed:
        If backtrack stack empty, entire match failed. EXIT.
        else pop new state off backtrack stack.
Wend.
Entire match succeeded. EXIT.
```

To make things clearer, special leaf nodes (`CUT`, `FAIL`, `NULL`), and the operators `~~`, `MANY`, and `MOST` weren't included in the algorithm. `~~` can be simply expressed in terms of backtrack- and call-stack actions: Push two special values onto the backtrack and call stacks. Attempt to match x. If x failed, the backtrack stack is popped. This indicates the negation succeeded. If x succeeded, the call stack is popped. This indicates negation failed, so pop all entries on the backtrack stack, including the special value, and continue backtracking from there.

`MANY` is defined in terms of `NULL`, `||`, and `&&`. A circular reference is used, since repetition is basically recursion. x `MANY` is equivalent to "match nothing, or else x followed by `x MANY`." Every failure following `x MANY` causes backtracking, which causes another repetition of x to be matched. The structure of `x MANY` (and of `x MOST`) appears in Figure Two.

Implementation

Since a major part of the string-matching algorithm is the case statement on the various node types, it makes sense to use data direction to implement it. The first cell (16 bits) of every pattern tree node points to the execution word for that routine. The node may have private data following. The stack action of the execution word is `( private -- flag )`, where private points to the start of private data and flag is true if the match succeeded.

For instance, the pattern `m' Hello'` returns adr, filling dictionary memory above adr as shown in Figure Three. Here, `<m>` is the address of the execution routine for `m'`. This is followed by the counted string to be matched. The pattern `m" abc" m" def" ||` sets up memory as shown in Figure Four (the details of the two subpatterns are suppressed for clarity). `<||>` is the execution word for alternation. This word performs the actions of an OR node, pushing the current state onto the backtrack stack and proceeding with the first option. The addresses of the two options are stored in the two private data words.

Most of the operators involved in pattern matching (`&&`, `||`, `~~`, `NULL`, `FAIL`, etc.) are unrelated to the representation used for strings. In fact, they can be used in any pattern-matching situation—for instance, matching lists. Hence, the pattern matcher is logically separable into two parts: a "logic engine" (LOGIC.SEQ) which provides the independent operators and the main loop, and a "string matcher" (STRMATCH.SEQ) which anchors patterns to the string by actually matching substrings.

The data-directed ap-

Figure Two.

Structure of
`x MANY` and `x MOST`

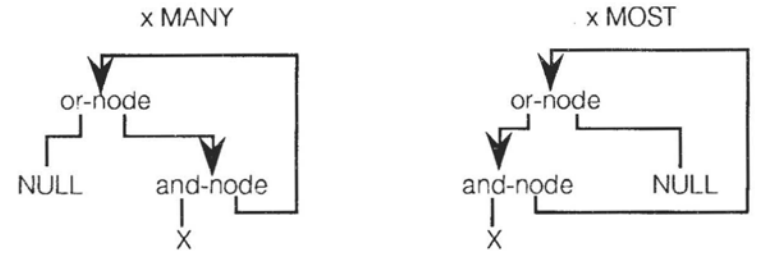


Figure Three.

Block returned by
`m' Hello'`

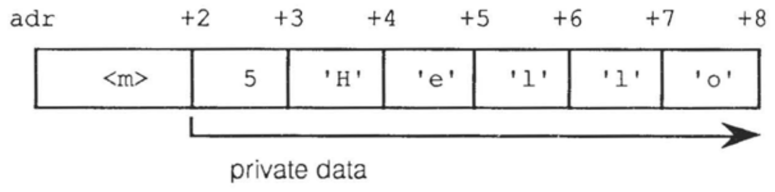
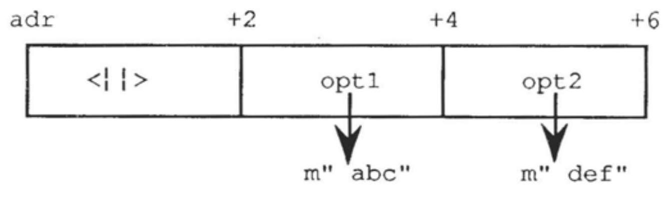


Figure Four.

Block returned by
`m" abc" m" def" ||`



proach allows us to define the main loop (in LOGIC.SEQ) before we define the string-matching nodes `M"` and `ANYOF"` (in STRMATCH.SEQ). It also leads to cleaner code: the main loop (`DRIVER` in LOGIC.SEQ) is defined by just 20 words, each operator is defined by two words (run time and compile time), and new operators are easily added.

Conclusions

Although I couldn't find any literature on the subject, I was able to implement advanced pattern matching by using standard Forth techniques: stacks, functional decomposition, minimal syntax, and separation of compile time from run time. Two complex designs led to two long, cumbersome, incorrect, and uncorrectable implementations. The final design is elegant, uniform, and easy to implement.

Data-directed programming is a very powerful technique which, in my opinion, isn't used enough. To use it, specify a uniform stack action for a class of routines and use their CFAs as type identifiers; then just `EXECUTE` your type identifier. The advantages: a `CASE` or `EXEC:` statement (and its overhead) is eliminated, and new data types can be installed transparently. `DRIVER` in LOGIC.SEQ doesn't "know" about any node types; it just knows the stack action. DDP isn't a substitute for OOP, but it's still useful!

Bibliography

I was unable to find a great deal of literature on string matching (unlike the related subject of parsing, which is very well established). What I could find, and used, was:

1. *Algorithms* (2nd ed.) by R. Sedgewick; Addison-Wesley, 1988, pp. 294-. Contains a chapter on matching regular expressions using finite-state machines.
2. *Vanilla SNOBOL4 — Tutorial and Reference Manual* by Mark B. Emmer. Distributed along with Vanilla SNOBOL4 by Catspaw, Inc. 1987. This is a public-domain implementation of the SNOBOL language with complete documentation. The documentation also mentions *The Macro Implementation of SNOBOL4* by Ralph Griswold (W. H. Freeman, 1972). I have been unable to find this book, but the description seems to indicate that it covers pattern matching.
3. Any UNIX system has AWK, LEX, and YACC, together with documentation. These programs are also available for the PC in various guises. AWK is a string-processing language, LEX is a lexical analyzer, and YACC is a parser generator used for writing compilers.

[Code implementing the ideas in this article will appear in our next issue. —Ed.]

Currently doing National Service, Ariel Scolnicov plans to start studying mathematics and computer science at Hebrew University next autumn. He enjoys Forth's ability to cover new concepts under a uniform syntax. Readers can correspond with him at P.O. Box 2747, Mevasseret Zion 90805, Israel.

Free! Trial Subscription

There are whole other worlds in micro computers than DOS and Windows. If embedded controllers, Forth, S100, CP/M or robotics mean anything to you, then you need to know about *The Computer Journal.*

Hardware projects with schematics, software articles with full source code in every issue. And you can try *The Computer Journal* without cost or risk! Call toll free today to start your trial subscription and pay only if you like it.

Rates: $18/year US; $24/year Foreign. You may cancel your subscription without cost if you don't feel The Computer Journal is for you. Published six times a year.

(800) 4248-TCJ / (908) 755-6186

TCJ The Computer Journal
The Spirit of the Individual Made This Industry
Socrates Press
PO Box 12
S. Plainfield NJ 07080-0012

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

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

MMSFORTH System Disk	from $179.95

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

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

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

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

MMSFORTH V2.4 System Disk	from $179.95

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

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

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

and a little more!

MMSFORTH

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

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

Call for free brochure, technical info or pricing details.

China's National Forth Examination

Translated by C.H. Ting
San Mateo, California

Last year, the People's Republic of China administered its first—and, as far as we know, the world's first—national test of Forth knowledge and expertise. In a huge country where positions in industry and academia are earnestly sought by many candidates, and where the results of formal examinations can be decisive factors shaping one's future, this test assumes great significance in the lives of Forth programmers there. We invite you to measure your own Forth expertise by taking the same examination. Those who would like to have their answers checked can send them to Dr. Ting at 156 - 14th Avenue, San Mateo, California 94402.

1991 China National Forth Programmer Examination

Total Time: 180 Minutes, Total Score: 180 points.

All problems are scored according to the Forth-83 Standard. Assume the base to be decimal unless noted otherwise.

Part One. Computer Fundamentals (36 points)

I. Calculations (Write down only the results.) (6 points)

1. Convert hex 1991 to binary. (2 points) ________
2. Convert hex ABE to decimal. (2 points) ________
3. Convert hex BAD to octal. (2 points) ________

II. Multiple Choices (26 points)

1. Convert a decimal fraction 0.875 to 8 binary digits. (6 points)
   Its octal expression is:
   A. 0.1100000 B. 1.1110000 C. 1.0100000 D. 0.0110000
   Its two's complement is:
   A. 1.0010000 B. 0.0100000 C. 1.1100000 D. 0.1100000
   Its one's complement is:
   A. 0.0011111 B. 1.0100000 C. 1.0001111 D. 0.0111111
2. The logic expression A(1+B) is: (2 points)
   A. A B. 1 C. AB D. $\overline{AB}$
3. A computer has a memory capacity of 64KB. Its address registers must have: (2 points)
   A. 15 digits B. 14 digits C. 16 digits D. 17 digits
4. The register which sequences the execution of Forth words is: (2 points)
   A. Work register W B. Program counter PC
   C. Interpreter pointer I D. Code field address CFA
5. In a CPU, the register which sequences the execution of instructions is: (2 points)
   A. Accumulator B. Program counter
   C. Internal address register D. Instruction register
6. A stack is a linear list. It is characterized by: (2 points)
   A. An end point B. A middle point
   C. First-in First-out D. First-in Last-out
7. The assembler producing object code for a different computer is called: (2 points)
   A. Macro assembler B. General assembler
   C. Micro assembler D. Cross assembler

HARVARD SOFTWORKS

NUMBER ONE IN FORTH INNOVATION

(513) 748-0390 P.O. Box 69, Springboro, OH 45066

MEET THAT DEADLINE ! ! !

- Use subroutine libraries written for other languages! More efficiently!
- Combine raw power of extensible languages with convenience of carefully implemented functions!
- Faster than optimized C!
- Compile 40,000 lines per minute! (10 Mhz 286)
- Totally interactive, even while compiling!
- Program at any level of abstraction from machine code thru application specific language with equal ease and efficiency!
- Alter routines without recompiling!
- Source code for 2500 functions!
- Data structures, control structures and interface protocols from any other language!
- Implement borrowed features, more efficiently than in the source!
- An architecture that supports small programs or full megabyte ones with a single version!
- No byzantine syntax requirements!
- Outperform the best programmers stuck using conventional languages! (But only until they also switch.)

HS/FORTH with FOOPS - The only full multiple inheritance interactive object oriented language under MSDOS!

Seeing is believing, OOL's really are incredible at simplifying important parts of any significant program. So naturally the theoreticians drive the idea into the ground trying to bend all tasks to their noble mold. Add on OOL's provide a better solution, but only Forth allows the add on to blend in as an integral part of the language and only HS/FORTH provides true multiple inheritance & membership.

Lets define classes BODY, ARM, and ROBOT, with methods MOVE and RAISE. The ROBOT class inherits:

```
   INHERIT> BODY
   HAS> ARM RightArm
   HAS> ARM LeftArm
```

If Simon, Alvin, and Theodore are robots we could control them with:

```
Alvin 's RightArm RAISE          or:
+5 -10 Simon MOVE                or:
+5 +20 FOR-ALL ROBOT MOVE
```

The painful OOL learning curve disappears when you don't have to force the world into a hierarchy.

WAKE UP ! ! !

Forth need not be a language that tempts programmers with "great expectations", then frustrates them with the need to reinvent simple tools expected in any commercial language.

HS/FORTH Meets Your Needs!

Don't judge Forth by public domain products or ones from vendors primarily interested in consulting - they profit from not providing needed tools! Public domain versions are cheap - if your time is worthless. Useful in learning Forth's basics, they fail to show its true potential. Not to mention being s-l-o-w.

We don't shortchange you with promises. We provide implemented functions to help you complete your application quickly. And we ask you not to shortchange us by trying to save a few bucks using inadequate public domain or pirate versions. We worked hard coming up with the ideas that you now see sprouting up in other Forths. We won't throw in the towel, but the drain on resources delays the introduction of even better tools that could otherwise be making your life easier now! Don't kid yourself, you are not just another drop in the bucket, your personal decision really does matter. In return, we'll provide you with the best tools money can buy.

The only limit with Forth is your own imagination!

You can't add extensibility to fossilized compilers. You are at the mercy of that language's vendor. You can easily add features from other languages to **HS/FORTH**. And using our automatic optimizer or learning a very little bit of assembly language makes your addition zip along as well as and often better than in the parent language.

Speaking of assembler language, learning it in a supportive Forth environment virtually eliminates the learning curve. People who failed previous attempts to use assembler language, often conquer it in a few hours using HS/FORTH. And that includes people with NO previous computer experience!

HS/FORTH runs under MSDOS or PCDOS, or from ROM. Each level includes **all** features of lower ones. Level upgrades: $25. plus price difference between levels. Source code is in ordinary ASCII text files.

HS/FORTH supports megabyte and larger programs & data, and runs as fast as 64k limited Forths, even without automatic optimization -- which accelerates to near assembler language speed. Optimizer, assembler, and tools can load transiently. Resize segments, redefine words, eliminate headers without recompiling. Compile 79 and 83 Standard plus F83 programs.

PERSONAL LEVEL $299.
NEW! Fast direct to video memory text & scaled/clipped/windowed graphics in bit blit windows, mono, cga, ega, vga, all ellipsoids, splines, bezier curves, arcs, turtles; lightning fast pattern drawing even with irregular boundaries; powerful parsing, formatting, file and device I/O; DOS shells; interrupt handlers; call high level Forth from interrupts; single step trace, decompiler; music; compile 40,000 lines per minute, stacks; file search paths; format to strings. software floating point, trig, transcendental, 18 digit integer & scaled integer math; vars: A B * IS C compiles to 4 words, 1..4 dimension var arrays; **automatic optimizer delivers machine code speed.**

PROFESSIONAL LEVEL $399.
hardware floating point - data structures for all data types from simple thru complex 4D var arrays - operations complete thru complex hyperbolics; turnkey, seal; interactive dynamic linker for foreign subroutine libraries; round robin & interrupt driven multitaskers; dynamic string manager; file blocks, sector mapped blocks; x86&7 assemblers.

PRODUCTION LEVEL $499.
Metacompiler: DOS/ROM/direct/indirect; threaded systems start at 200 bytes, Forth cores from 2 kbytes; C data structures & struct+ compiler; MetaGraphics TurboWindow-C library, 200 graphic/window functions, PostScript style line attributes & fonts, viewports.

ONLINE GLOSSARY $ 45.

PROFESSIONAL and PRODUCTION LEVEL EXTENSIONS:

FOOPS+ with multiple inheritance **$ 79.**
TOOLS & TOYS DISK $ 79.
286FORTH or 386FORTH $299.
16 Megabyte physical address space or gigabyte virtual for programs and data; DOS & BIOS fully and freely available; 32 bit address/operand range with 386.
ROMULUS HS/FORTH from ROM **$ 99.**

Shipping/system: US: $9. Canada: $21. foreign: $49. We accept MC, VISA, & AmEx

8. An operating system improves the _______ of a computer. (2 points)
   A. Speed B. Utility of resources
   C. Flexibility D. Compatibility
9. In an operating system, which component is responsible for thecontrol of process execution? (2 points)
   A. Main memory manager B. Microprocessor manager
   C. Processor manager D. Page memory manager
10. The component controlling the data in a system is: (2 points)
   A. Data manager B. Document manager
   C. Indexing system D. Data storage system
11. The most obvious difference between Forth and other computer languages is: (2 points)
   A. It has the functions of an operating system.
   B. It is an integrated programming environment.
   C. It has a dictionary with stack structure.
   D. It improves the portability of programs.

III. Mark a correct statement with "O", and an incorrect one with "X". (4 points)
   A. A computer without external components is a 'Bare Computer'. __________
   B. Converting often used software operations to hardware can improve the efficiency of a computing system. ___________

Part Two. Forth Fundamentals (90 points)

I. Fill the blanks (25 points)

1. ______________, ______________, _____________ are the three internationally recognized Forth standards. (3 points)
2. The postfix expression of [(A-B)*C]/(D+E) is
```
A B - C _____ E _____ /                        (3 points)
```
3. To reorder the stack (1 2 3 4 -- 2 1 4 3), use:
```
SWAP __________ SWAP __________                (2 points)
```
4. Compute the factorial of n (n -- n!), with n>1:
```
: N! ______ DUP ROT ?DO I * -1 +LOOP ;         (1 point)
```
5. Define Y= to compute y= ax^2+bx+c, with the stack changes as shown:
```
: Y=  ( x a b c -- y )
      >R 2 PICK * >R  ________ DUP * ________ ( ax*x )
      R> R>   + + ;                            (2 points)
```
6. 3DUP (n1 n2 n3 -- n1 n2 n3 n1 n2 n3) is equivalent to
________ ________ _________ . Each blank can only hold one Forth word. (3 points)
7. Convert a double integer to its absolute:
```
: DABS ( d -- |d| )
      DUP 0<   IF _____________ THEN ;         (1 point)
```
8.
```
: TEST   2. 0 -   IF 1 ELSE 2 THEN
      DUP . =   IF 3 . ELSE 4 . THEN ;
Typing TEST <cr> will display _____________    (1 point)
```
9.
```
DECIMAL 16 BASE !   BASE @ . <cr> will display:
___________                                    (1 point)
```
10.
```
HEX   : NUM   DECIMAL 10 HEX 10 + . ; DECIMAL
Type NUM <cr> will display ___________         (1 point)
```
11.
```
32767 1+ . <cr> will display ___________       (1 point)
```
12.
```
-1 1234 C! 1234 C@ . <cr> will display _______ (1 point)
```
13.
```
2VARIABLE XY   3 5 XY 2!
XY @ . < cr> will display ___________          (1 point)
```
14.
```
1 NOT . <cr> will display ___________          (1 point)
```
15.
```
HERE . <cr> shows 3000
: TEST 500 CR ;
HERE . <cr> will show ___________              (1 point)
```
16.
```
HERE . <cr> shows 3000
1 , 2 C, HERE . will show ___________          (1 point)
```
17. Following is the incomplete definition of DO. It compiles (DO) into the dictionary and leaves HERE and 3 on the stack. Complete definition:
```
: DO COMPILE (DO) HERE 3 ; _______________     (1 point)
```

II. Multiple Choices (25 points)

1. The inventor of Forth is:
   A. English B. American C. Swiss D. Japanese

```
2.     Forth originated in:
       A. Chemical industry          B. Automotive industry
       C. Astronomy                  D. Light manufacturing
3.     The postfix expression of 15/(7-2) is
       A. 15 / (7-2)                 B. 15 7 / 2 -
       C. - 7 2 / 15                 D. 15 7 2 - /
4.     : TEST ( ABC ." DEF" ) ;
       Type TEST <cr> will display:
       A. ABC       B. DEF           C. ABCDEF        D. nothing
5.     In the stack comment ( n addr c -- f )
       n is for an
       A. Integer   B. Address       C. Character     D. Logic flag
       f is for an
       A. Integer   B. Address       C. Character     D. Logic flag
6.     DECIMAL : TEST 16 HEX 10 * * ;
       Type TEST <cr> will display
       A. 16 10     B. A 16          C. A 10          D. 10 A
7.     In Forth-83 Standard, the range of ud is:
       A. 0 to 65535                 B. -32768 to 32767
       C. 0 to 4294967295            D. -32768 to 65535
8.     In Forth-83 Standard, the range of addr is:
       A. -32768 to 32767            B. -128 to 127
       C. -32768 to 65535            D. 0 to 65535
9.     The result of 13 -7 MOD is:
       A. 5          B. -5                C. 1             D. -1
10.    0 -1 DABS D. will produce:
       A. -1         B. 0                 C. 65536         D. 65535
11.    -18 5 / will produce:
       A. 4          B. 3                 C. -4            D. -3
12.    2 NOT 3 + -1 AND will produce:
       A. -1         B. 0                 C. 4             D. 1
13.    Reorder the stack ( a b c d -- d c b a )
       A. ROT ROT ROT ROT
       B. SWAP 2SWAP SWAP 2SWAP
       C. SWAP 2SWAP SWAP DUP DROP
       D. 2SWAP SWAP 2SWAP SWAP
14.    : TEST CR   3 1 DO   6 4 DO   J 0 .R I 0 .R 5 SPACES
              LOOP CR LOOP ;
       Typing TEST <cr> will display:
       A. 14     15                  B. 41     51
          24     25                     42     52
       C. 14     15     16           D. 41     51     61
          24     25     26              42     52     62
          34     35     36              43     53     63
15.    : TEST 0 1 5 DO   I + -2 +LOOP . ;   TEST will produce:
       A. 15    B. 14                C. 8             D. 9
16.    : TEST 1000 = IF KEY DROP EXIT
              ELSE HEX THEN ;
       The total length of this word in the dictionary is:
       A. 33 bytes    B. 29 bytes C. 31 bytes     D. 35 bytes
17.    VARIABLE XX 4 ALLOT   VARIABLE YY 0 ,
       ' YY >NAME   ' XX >NAME - . <cr> will display:
       A. 0           B. 9         C. 13            D. 11
18.    CR 3 SPACES 32 EMIT   OUT @ 0 .R   OUT @ 0 .R <cr> will display:
       A. 43          B. 34        C. 45            D. 54
19.    HERE . <cr> shows 5000
       HEX 10 ALLOT DECIMAL HERE . <cr> will display:
       A. 5016        B. 5000      C. 5010          D. 5020
20.    To compile an immediate word into a colon definition, use:
       A. COMPILE    B. [COMPILE]  C. ]             D. IMMEDIATE
21.    To create a header for XXX and point DP to the beginning of its parameter field, use:
       A. BUILDS> XXX                     B. CREATE XXX
       C. : YYY CREATE XXX ;              D. : XXX ;
22.    Define the following word:
       : XXX CREATE , DOES> @ ;
```

```
        Execute 176 XXX YYY
        A. A new word YYY is compiled to dictionary. Stack has 176.
        B. YYY is compiled to dictionary.  176 is put in its parameter field.
        C. YYY is compiled to dictionary, 176 is put in its parameter field, and stack has the
           parameter field address.
        D. No new word is compiled to the dictionary.
23.     To compile 64*3 as a literal in a colon definition, use
        A. LITERAL 64 3 *                 B. [LITERAL] 192
        C. [ 64 3 * ] LITERAL             D. 64 3 *
```

```
III.    Determine Errors (20 points)
1.      Define two vocabularies to be used in the subsequent definitions:  (10 points)
        VOCABULARY XX         VOCABULARY YY        YY DEFINITIONS
               A                    B                    C
        : ABC ;               XX DEFINITIONS       : DEF ;
           D                        E                  F
        : GHI    YY DEF     XX ABC   ;
           G       H          I      J
        There are ________ errors, identified by _________________.
2.      Define a variable BASE1 and use it as the base to display a number n. (10 points)
        : TEST ( n -- )       BASE @        R>       VARIABLE BASE1
                                A           B              C
               BASE1 @        BASE !        0.       <# # # # # #>
                  D             E           F              G
               TYPE           >R            BASE ! ;
                 H            I                 J
        There are ________ errors, identified by _________________.
```

```
IV.     Stack Analysis ( 20 points)
        Analyze the following program and trace the stack. Use the standard stack notation in the
        comment parentheses.
        DECIMAL
        : #IN ( -- n , push the number entered on keyboard on the stack)
        0 BEGIN       KEY              ( n c                          )
                      DUP 13 =         ( n c f                        )
                      IF DROP          (                    1         )
                      EXIT
                      THEN
                      DUP 8 =          (                    2         )
                      IF EMIT 32 EMIT 8 EMIT
                      10 /             (                    3         )
                      ELSE DUP         (                    4         )
                      48 <             (                    5         )
                      OVER             (                    6         )
                      57 >             (                    7         )
                      OR               (                    8         )
                      IF DROP 7 EMIT9
                      ELSE DUP EMIT
                      48 -             (                    9         )
                      SWAP 10
                      * +              (                    10        )
                      THEN
                      THEN
        AGAIN ;
```

Part Three. Program Design (36 points)

```
I.      Fill in Forth Words (22 points)
        In the following program, fill in the appropriate Forth words in the blank fields. WORDS is
        used to inspect the contents of the current vocabulary.
        : WORDS HEX CR CR
        ( find the name field address of the last words in the currentvocabulary)

        ________________________
                ?DUP IF BEGIN DUP DUP 0
        <# # # # # #>
        TYPE IF  ( display a blank character) _____________
        ( show the name of a word ) _____________
        ELSE  ( display "Null" ) ____________________________ DROP
```

```
THEN OUT @ 30 >
IF ( start a new line) ___________
ELSE 14 OUT @ OVER MOD -
( display that many blank characters) ______________
THEN
( from name field find the name field address of the next word)
________________________________ DUP 0 =
( is there any keyboard activity? ) ______________
DUP IF ( wait for the key ) ______________ DROP THEN
( is any one of the two flags true? ) ____________ UNTIL
ELSE ( display "Empty vocabulary") _________________________
THEN ;
```

II. Program Design (14 points)

The equations to draw a circle are:

X= x0 + rCOSa and Y= y0 + rSINa

where (x0,y0) are the center coordinates, r is the radius, and a is an angle. Assume that we have a word LINE (x1 y1 x2 y2 color --) which draws a line from (x1,y1) to (x2,y2). Color 0 is the background color and color 1 is white.

Write a program to draw circles. First compute one point on the circle (x1,y1) as the starting point. Then compute (x2,y2) and draw a line between these two points. Make (x2,y2) the (x1,y1) of the next segment. Compute the next point and draw the next segment. Increment a to compute the next point. Increase a from 0 to 360 degrees and draw the complete circle.

SIN (deg -- sine*10000) and COS (deg -- cosine*10000) are predefined words which convert angles to sines and cosines multiplied by 10000 so that they are represented by integers. Avoid integer overflows in the program. (Note: the coordinates of the end point can be saved as double integers to initialize the starting coordinates of the next segment).

Define the following words:

```
X= ( x0 r a -- x )
Y= ( y0 r a -- y )
XY= ( x0 y0 r a -- x y )
CIRCLE ( x0 y0 r -- )
```

Define each word separately and write a complete circle drawing program.

Part Four. English (18 points)

I. Select the most appropriate words in the list to fill in the blank spaces in the next three paragraphs. (9 points)

1. Forth is ________________ . Because you can add to the language, you can tailor it to your own needs. Since almost everything in Forth is written in Forth--the text editor, assembler, etc.--you can access and alter all of it.

2. Forth is ________________ . Forth runs much faster than many other high-level languages. Because the interpretation scheme is so elegant, interpreter overhead is minimal. Furthermore, Forth includes a built-in assembler for speed-critical routines. As a result, Forth can run almost as fast as machine code itself.

3. Forth is ________________ . Only a small nucleus of code needs to be rewritten to move the entire language to a new computer. Forth has been implemented on almost every computer developed to date.

Answers to be selected:

A. fast	B. slow	C. compact	D. powerful
E. extensible	F. transportable	G. interactive	H. structured
I. restrictive			

II. Translate the following paragraphs into Chinese (9 points)

1. Forth is a language, an operating system, a set of tools, and a philosophy. It is an ideal means for thinking because it corresponds to the way our minds work. <Thinking Forth> is thinking simple, thinking elegant, thinking flexible. It is not restrictive, not complicated, nor over-general. <Thinking Forth> synthesizes the Forth approach with many principles taught by modern computer science. (3 points)

2. Business, industry, and education are discovering that Forth is an especially effective language for producing compact, efficient applications for real-time, real-world tasks.

<Thinking Forth> combines the philosophy behind Forth with thetraditional, disciplined approaches to software development--to give you a basis for writing more readable, easier-to-write, and easier-to-maintain software applications in any language. (3 points)

3. Forth is like the Tao: it is a way, and is realized when followed. Its fragility is its strength, its simplicity its direction.

A Forum for Exploring Forth Issues and Promoting Forth

Fast FORTHward

Promoting Forth and Other High-Tech Stuff

To prepare a press "backgrounder" for Forth, I followed some simple procedural steps. Similar steps may help you to promote other high-tech products or services. As part of this exploration, I will be sharing backgrounder excerpts explaining various advantages of Forth.

To help generate promotional copy for Forth, my first step was to develop a brief list of Forth's advantages. Next, I investigated which advances could best summarize what Forth is all about.

The focus then became a few Forth advantages around which I tried to create stories with *promotional* messages. After several stories were compiled, the stories were tied together as much as possible.

Top on my list of advances was Forth's extensibility and scalability. While the ability to extend a language is frequently mentioned, the ability to scale a language is rarely touted.

A term similar to scalable is *open*. Openness also characterizes Forth well. To suit the length restrictions of an advertisement, the following *short story* was used:

To use this message in advertising, it must be made simpler.

> As an *open* language, Forth lets programmers build new control-flow structures and other compiler-oriented extensions that closed languages do not.

Another advantage I noted was Forth's usefulness as a meta-language suitable for creating application-specific languages.

But Forth's linguistic flexibility is not characterized by merely describing Forth as *scalable* instead of *extensible*—nor by describing Forth as a meta-language. Despite how deeply Forth programmers appreciate this message, it can't be deciphered by those who have never used Forth.

Perhaps the difficulty of this Forth concept is due to the terminology required for its expression. This prompted me to take a fresh approach to the telling of this Forth story by avoiding terms such as *grammar*. Here is how it turned out:

> In most languages, the declaration of data items or routines helps enrich the variety of useful expressions that can be programmed. The appeal of most languages arises due to this one trait—extensibility in the domain of expressions. It allows one or more programmers to build layered, modular applications.
>
> Forth has improved upon the best trait of other languages—widening expressions until they engulf the Forth language as a whole: All the elements of the Forth language correspond to expression constructs, with even fewer syntax rules than is customary for expressions.
>
> No irreversible Forth grammar or syntax is needed to ensure that programmers specify routines or variables only in places where they are *permissible*. Forth's lack of type-checking contributes to this freedom.
>
> By eliminating the need to nest expressions within parentheses, Forth's postfix notation avoids still other (syntax) constraints—those involving correctly paired, and correctly placed, parentheses within expressions. When expressions are nonstop as they are in Forth, the phrases "in an expression" and "nested expression" lose all their meaning.

Other languages continue to limit extensibility to the domain of expressions. Furthermore, the permissible components of expressions are also limited—just try placing an IF in the expression portion of a PRINT statement in BASIC.

> Due to its abandonment of left-to-right evaluation, algebraic notation mandates your use of correctly paired parentheses as necessary to specify how the result of an inner expression should be passed to an outer expression. The order of evaluation proceeds in an inside-out fashion. For reasons unknown to most Forth programmers, this notational sequence is considered easier and more readable. (Nevertheless, microprocessors require the re-ordering of equivalent machine code to reflect its real execution order.)

Terms such as grammar and semantics lead to more conventional ways of expressing Forth's flexibility, but they also raise the discussion to a higher technical level:

> Unlike most languages, Forth provides a measure of linguistic self-determination to its users. Normally, components of

languages are under the strict control of compiler vendors. Through its support of arbitrary application grammars, Forth lets the programmer determine the grammar and semantics suited to a given project. Accordingly, the language for an application can truly be fitted to the application. This flexibility has been designed into Forth.

The *switch* statement of the C programming language is useful for programming in terms of state machines. While users of conventional languages other than C have to wait for the various compiler vendors to adopt this new language construct, Forth programmers have remained free to add components such as a *switch* to virtually every implementation of Forth that has ever been created.

Because of its length and complexity, this particular Forth message remains unwieldy. To use it in an advertisement, it must be made simpler still.

Let's continue to refine this message regarding Forth's flexibility. Perhaps you can share some of your own recipes for the promotion of Forth. Together, let's make Forth a poorly kept secret one day. —*Mike Elola*

(un)Vendor Spotlight

Forth Interest Group Update

At a recent planning meeting, the FIG Board of Directors along with a few volunteers undertook the task of evaluating where it needs to focus its efforts. Besides a mission statement, the planners brainstormed to produce a list of about 40 possible FIG activities that could help fulfill FIG's mission. Evaluating those activities took the form of rating how much each of those activities supported each of the objectives in the mission statement.

The planners also studied the organizational structure of FIG, producing an "org" chart to help formalize each of the roles being performed by various FIG supporters. At the same time, we identified the need to fill several vacated, or under-emphasized roles in the organization.

Since then, FIG president John Hall has made several important appointments: John Rible is the FIG Chapter Coordinator; Nick Solntseff is the Education Coordinator; and Mike Elola is the Publicity Director. John may appoint another volunteer as merchandiser of the FIG mail-order business.

Nick, John, and Mike are charged with helping educate Forth programmers, supporting chapters, and promoting Forth and FIG, respectively.

The use of "task champions" is making FIG more responsive to new ideas, such as the column you are reading now. If you wish to help in any of the task areas outlined, send us a brief description of your interests, listing any professional skills you have so we can build a talent-pool database. The FIG office will forward any specific comments or suggestions you make to the appropriate task champion.

Product Watch

April 1992

Orion Instruments announced the PC-based 8800 emulator/analyzer. It has the speed needed to make it a real-time, zero wait-state emulation of Motorola's 68000 and 68302 microprocessors running at well over 40 MHz. (Support for 80C196, 68332, and 68HC16 is planned.) Besides allowing source-level debugging, the 8800 can use the host PC's 386 protected mode to run other programs, such as editors and compilers. By placing such programs in the "User" menu, all such programs can become an integrated part of the 8800 operating environment.

Also announced as an option for the 8800 is Clip-On™ Emulation. It ensures that timing is unaffected by the emulator.

April 1992

Vesta Technology announced the Vesta SBC332, a low-cost, high-speed, low-power, single-board computer based on the Motorola MC68332 microcontroller. A single unit may be obtained *free* with the purchase of an SDSI debugger, or with the purchase of chipFORTH332™ from Vesta Technology. (Forth, Inc. is responsible for chipFORTH332 and trademarks that name.)

Several add-on products are available, such as one to augment the SBC332 with serial I/O and A/D as well as D/A conversion (the MFP332 Multi-Function Peripheral).

Also available now in a version for the SBC332 is the Vesta Standard Edition, a 32-bit subroutine-threaded Forth.

Companies Mentioned

Orion Instruments
180 Independence Dr.
Menlo Park, California 94025
Fax: 415-327-9881
Phone: 415-327-8800

Vesta Technologies
7100 West 44th Ave., Suite 101
Wheat Ridge, Colorado 80033
Fax: 303-422-9800
Phone: 303-422-8088

CALL FOR PAPERS

for the fourteenth annual

FORML CONFERENCE

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

Following Thanksgiving
November 27 – November 29, 1992

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

Theme: Image display, capture, processing, and analysis

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

Mail abstract(s) of approximately 100 words by September 1, 1992 to FORML, P.O. Box 2154, Oakland CA 94621.

Completed papers are due November 1, 1992.

Registration information may be obtained by telephone request to the Forth Interest Group (510) 893-6784 or by writing to FORML, P.O. Box 2154, Oakland, CA 94621.

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

This conference is sponsored by FORML, an activity of the Forth Interest Group. Information about membership in the Forth Interest Group may be obtained from the Forth Interest Group, P.O. Box 2154, Oakland, CA 94621, telephone (510) 893-6784.

Best of GEnie

Gary Smith
Little Rock, Arkansas

There will be very little preamble in this edition of 'on-line' notes. The contents of the messages speak for themselves. The first group was pulled from Category 1, Topic 6, "Real-Time Conferences," and discusses our disheartening decision to terminate the Thursday Figgy Bar. The second is captured from Category 10, Topic 24, "ANS TC Magnet for Interpreter" and features a lively exchange on GEnie, and from throughout ForthNet, regarding a hypothetical use of local/user variables.

Topic 6: Real-Time Conferences

Real-Time Forth Conferencing on this RoundTable

The Thursday night Figgy Bar will cease, except for the monthly guest appearance after the February 6th session. Attendance has fallen beyond acceptable levels for more than a month. If you wish to preserve this on-line chapter meeting and if you have any opinions this is the time to share them.

—Gary Smith

From: Dennis Ruffer

Well, we had our final Thursday night Figgy Bar last Thursday night, and I must say, it doesn't look like any of you will miss them. No one but us sysops showed up for the final tribute. In fact, no one has even bothered to comment, one way or the other, about this loss. This is disappointing to those of us who have gone out of our way to make these happen for you. We started these Thursday night get-togethers for you to share your thoughts about Forth projects you are working on and to hear what others are doing. The intention was to provide an on-line FIG chapter meeting for those of you who do not often get the opportunity to talk with other Forth programmers.

We started with a good core group of people who would show up frequently. I'd like to take this opportunity to thank them for their time and enthusiasm. They made these conferences enjoyable and helped us keep them going for the past two years. Unfortunately, everyone's priorities change over time, and this core group dwindled down to nothing. Most recently, Gary has been frequently sitting in the RTC completely alone. This is most disheartening for him, after he has gone to the (sometimes extraordinary) effort to be there, and few (if anyone) even shows up. Now, after much discussion, there just is no desire (on any of our parts) to continue making the effort any more.

This does not mean they will not come back, but you the user must show us that you want us to do it. We are here for your benefit, but if you do not choose to take advantage of something we are doing, we will shift our efforts to something else that you might find useful. However, this means you must tell us what you want. If the time was not right, tell us so. If you'd like us to use the on-line RTC rooms in some other ways, let us know what you want. If someone out there would like to volunteer to help us out, please step forward. We need ideas, and enthusiastic users and volunteers. Please contribute wherever you can. Otherwise, as has happened with the Thursday night Figgy Bars, the benefits will just disappear.

This brings us to the guest conferences. Speakers Gary lined up recently include:

2/20—Paul Thomas of Sun Microsystems and co-author with prior guest Gary Feierbach of *Forth Tools and Applications*. Paul's topic: "Using CREATE ... DOES>."

3/19—Ron Braithwaite, independent consultant. Ron's topic: "Professional Impact of dpANS Forth."

4/16—Len Zettel. Len's topic: "Is the Forth Community Missing the Boat?"

However, attendance at these conference have also been disappointing and very embarrassing for us and the guest speaker. Not only has Gary spent hours on the phone lining up these guests, but the guests themselves have taken time out of their busy schedules to talk to us. Then, when no one shows up, everyone goes away disappointed. Obviously, under those conditions, guests rarely want to come back again, and finding new guests becomes harder. Again, the enthusiasm is dwindling and it is questionable how much longer we will continue to have them. There is, on the other hand, still an opportunity for you to show us how much these are worth to *you*. You only have to show up and join the discussion to show us you care. If these next three go well, we will schedule more. Otherwise, these too will become a thing of the past.

ANS Forth Interpreter

ANS Forth Standard for Interpreter. Magnet: Dean Sandersen

From: Sabbagh

Subject: `: Pt+ ( pt1 \ pt2 -- pt3 )`

Schmidtg@iccgcc.decnet.ab.com writes:

"Hadil G. Sabbagh writes:

"'Let's consider the following problem: a point is represented on the stack as integers x y z. I wish to define ^ ^ ^

"'Note that a point as defined here consists of three coordinates. Again, note how the problem has changed to points with two coordinates!'

"Finally, Hadil G. Sabbagh writes:

"'...where `in1`, `in2`, ... are input arguments; `l1`, `l2`, ..., are locals; and `out1`, `out2`, ... are output arguments. Thus, we have

```
    : pt+ { x1 y1 x2 y2 --> x3 y3 }
        x1 x2 + -> x3
        y1 y2 + -> y3  ;
```

"Now the original poster is convinced that he is solving a different problem!"

Just going with the flow, Greg. Here is my original problem (and solution, using locals).

```
: pt+   { x1 y1 z1 x2 y2 z2 --> x3 y3 z3 }
  x1 x2 + -> x3
  y1 y2 + -> y3
  z1 z2 + -> z3  ;
```

See my previous posting on the locals syntax.

—Hadil

From: Bernd Paysan
Sabbagh writes:

"Just going with the flow, Greg. Here is my original problem (and solution, using locals).

```
      : pt+ { x1 y1 z1 x2 y2 z2 --> x3 y3 z3 }
            x1 x2 + -> x3
            y1 y2 + -> y3
            z1 z2 + -> z3  ;
```

"See my previous posting on the locals syntax.
"—Hadil"

I wrote some weeks ago a library for vector arithmetics (not only addition and subtraction, but dot and cross product and 3d>2d projection). I solved it with an array of six floats (called temp). This is a dirty solution, but it works well:

```
CAPS OFF
\ case insensitive: writes faster, reads better
Create temp 6 floats allot DOES>
swap floats + ;
: temp!  5 FOR  i temp f!
  NEXT ;
: v+ ( pt1 pt2 -- pt3 )
  temp!  3 0 DO  i temp f@
  i 3 + temp f@ f+ LOOP ;
```

(and so on).

If I had a FP unit, I would use its registers as temporary and win lots of time. The temp array is useful for swap, dup, dot, and cross products, for all the things you do with three-dimensional vectors. The order of the vector on stack (x,y,z) or (z,y,x) is only important for the cross product (left hand or right hand). I like the (x y z --) format because you type it in this order. I suppose you do the same, but then tell me one thing: How does your Forth get the locals in the right order? Every locals I know turn the order round and you have C conventions to give parameters, i.e.:

```
: GCT ( a b -- gct ) \ has locals
  { b a } ... ;
```

—Bernd Paysan

From: Jan Stout
Doug Philips writes:

"Interesting. I have an entirely opposing view. If one is trying to use Abstract Data Types (ADTs), then one doesn't want to see the 'exploded' stack diagrams in the documentation for the ADT's *primitive* operations. If that was done then the DT is hardly A!"

Hm, my view is that the user of the ADT shouldn't see the definition of the ADT at all. In other languages this is achieved by a double specification, one on the interface level (that would use pt1, pt2, etc.) and one on the implementation level giving away the internal representation (i.e., pt1x pt2y).

Applied to Forth, I find the (word) name descriptive enough for interface level, so the pt1 pt2 wouldn't show up at all…

"I think the *real* problem is that there is no easy way in Forth to deal with small structures passed directly on the stack. Using an array in this case (two or three dimensions) seems like overkill. However, I'm not at all impressed with the alternative above."

I've come up with the following stack juggling that seems to call for a factor (just how would we name such a beast?):

```
: Pt+ ( x0 y0 x1 y1 -- x y )
  >R  ROT +  R>  ROT + ;
```

Not convinced? Well see how "easily" it's generalized to the three-dimensional problem:

```
: Pt+ ( x0 y0 z0 x1 y1 z1 -- x y z )
  >R >R  3 ROLL +  R>  3 ROLL +  R>  3 ROLL + ;
```

Well that makes 0.04 doesn't it?

From: Doug Philips
Jan Stout writes:

"Doug Philips writes:
"'Interesting. I have an entirely opposing view. If one is trying to use Abstract Data Types (ADTs), then one doesn't want to see the 'exploded' stack diagrams in the documentation for the ADT's *primitive* operations.'
"Hm, my view is that the user of the ADT shouldn't see the definition of ADT at all. In other languages this is achieved by a double specification, one on the interface level (that would use pt1, pt2, etc.) and one on the implementation level giving away the internal representation (i.e., pt1x pt2y)."

I'm in agreement so far.

"Applied to Forth, I find the (word) name descriptive enough for interface level, so the pt1 pt2 wouldn't show up at all…"

Okay. I was considering the "standard practice" of having the first line of a definition, which usually contains the word's overall stack diagram, to be a word's documentation. But I agree, the documentation shouldn't divulge the details of the implementation. Which leads to an interesting question: How do you (if you do) avoid having to:

a) know how many cells a Pt uses *or*

b) have to invent a gazillion words Pt>R, R>Pt, PtSWAP, PtROLL…

"Well see how 'easily' it's generalized to the three-dimensional problem:

```
      : Pt+ ( x0 y0 z0 x1 y1 z1 -- x y z )
        +   >R >R  3 ROLL +  R>  3 ROLL +
        R>  3 ROLL + ;
```

"Well that makes 0.04 doesn't it?"

Unfortunately, I don't buy it. The sequence 3 ROLL + doesn't tell the whole story. The sequence R> 3 ROLL + is a better candidate except for the first usage. How about:

```
: Pt+ ( x0 y0 z0 x1 y1 z1 -- x y z )
  >R >R >R
  R> 3 ROLL +    R> 3 ROLL +    R> 3 ROLL + ;
```

And let the compiler optimize out the >R R> pair?

Or perhaps

```
: (Pt+)  COMPILE R> COMPILE 3 COMPILE
  ROLL COMPILE + ; IMMEDIATE
: Pt+ ( x0 y0 z0 x1 y1 z1 -- x y z )
  >R >R >R   (Pt+) (Pt+) (Pt+) ;
```

But I don't much like that either.

—Doug

From: Lennart Staflin

Bernd Paysan said:

"I suppose you do the same, but then tell me one thing: How does your Forth get the locals in the right order?"

With the syntax: `LOCALS| a b c |`

The following definition would do:

```
: LOCALS|
  \ Compiling:  ( "l1 l2 ... ln <|>" -- )
  \ Run time:   ( x1 x2 ... xn -- )
  0   ( End marker )
  BEGIN  BL PARSE-WORD   ( a # )
    OVER C@ [CHAR] | =
  UNTIL              ( 0 a1 #1 a2 #2 ... an #n )
  BEGIN  DUP WHILE  (LOCAL)  REPEAT      ( 0 )
  0 (LOCAL)  ;  IMMEDIATE
```

—Lennart Staflin

There's more to life than books, you know...
but not much more. —The Smiths

From: Eric S Johansson

Schmidtg@iccgcc.decnet.ab.com writes:

"'Instead of locals, another possibility has appeared in the press. That is a mechanism for specifying before/after stack pictures which cause stack transformation at run time. Typically it looks something like:

`( sn abcdef --- facdbe )`

"'where the "abcdef" is the before picture and the "facdbe" is the after picture. The two major problems with this technique are implementation, and notation.'

"When I read about this in *FD*, it seemed like a workable scheme, but interpreting strings to effect stack transformation does not strike me as an elegant (read Forth-like) solution to the problem."

Interesting point. Maybe what we are discusing here is really about the most "Forth-like" way to handle stack manipulations when a data element may consist of more than one stack cell (i.e., a point in three-space or an RGB triplet pixel).

—Eric

Source of the public's fear of the unknown since 1956.

From: Bernd Paysan

Lennart Staflin writes:

"Bernd Paysan said:

"'I suppose you do the same, but then tell me one thing: How does your Forth get the locals in the right order?

"'With the syntax: `LOCALS| a b c |`

"'The following definition would do:

```
    : LOCALS|
    \ Compiling:  ( "l1 l2 ... ln <|>" -- )
    \ Run time:   ( x1 x2 ... xn -- )
    0   ( End marker )
     BEGIN  BL PARSE-WORD   ( a # )
      OVER C@ [CHAR] | =
    UNTIL          ( 0 a1 #1 a2 #2 ... an #n )
     BEGIN  DUP WHILE  (LOCAL)  REPEAT ( 0 )
    0 (LOCAL)  ;  IMMEDIATE
```

Thanks. Looks genial. I have two problems: First the block or the TIB may change its location between parsing (think of multi-programming systems). Second, I can't create headers given a string (I can, but it is difficult). I found a solution: I stack the contents of `>IN`, and everything works well. I have four words for creating locals: `<LOCAL` starts the definition, `LOCAL: <name>` creates a local, `LOCAL>` ends the definition and `LOCAL;` ends scope of locals. All these words are immediate, so they can work without adding new definitions. So the solution is:

```
: {  POSTPONE <LOCAL  -1
( end marker, >IN will never be -1)
  BEGIN  >IN @ BL WORD 1+
  C@ [CHAR] | =  UNTIL  DROP >IN @ >R
  BEGIN  DUP 0< 0=  WHILE  >IN ! POSTPONE
  LOCAL:  REPEAT  DROP
  POSTPONE LOCAL> ; IMMEDIATE

: } POSTPONE LOCAL; ; IMMEDIATE
: TEST { A B | A . B . } ;
```

and

```
1 2 TEST
```

gives

```
1 2  ok
```

Hurray!

—Bernd Paysan

From: Jan Stout

Doug Philips writes:

"...question: How do you (if you do) avoid having to:

"a) know how many cells a Pt uses OR

"b) have to invent a gazillion words `Pt>R`, `R>Pt`, `PtSWAP`, `PtROLL`...

"Unfortunately I don't buy it. The sequence `3 ROLL +` doesn't tell the whole story. The sequence `R> 3 ROLL +` is a better candidate except for the first usage. How about:

```
    : Pt+  ( x0 y0 z0 x1 y1 z1 -- x y z )
      >R >R >R   R> 3 ROLL +   R> 3 ROLL +   R>
    3 ROLL + ;
```

"And let the compiler optimize out the >R R> pair?"

Sure.

"Or perhaps

```
    ": (Pt+)  COMPILE R> COMPILE 3 COMPILE ROLL
    COMPILE + ;
    IMMEDIATE"
```

Why go immediate?

```
: 3dCoord+   ( n0 n1 n2 n3  -- n1 n2 n4 )
  3 ROLL + ;
```

```
    ": Pt+ ( x0 y0 z0 x1 y1 z1 -- x y z )
        >R >R >R    (Pt+) (Pt+) (Pt+) ;"

: 3dPt+ ( x0 y0 y1 x1 y1 z1 -- x y z )
  >R >R >R   R> 3dCoord+  R> 3dCoord+  R> 3dCoord+ ;
```

"But I don't much like that either."

Well, I suppose dumpin' the lot on an array and referencing the coordinates from there *would* be more readable/ efficient...which brings up my following question:

Is the following allowed in the current LOCALs scheme?

```
: <    ( n m -- ? )
  LOCAL m    m -   0< ;
```

(Thus mixing implicit and explicit parameter passing).

If it were, that would disallow the very efficient implementation of just keeping the locals on stack till the ; where the upper stack part would be shifted down #locals-used times.

—Jan Stout

From: John Wavrik

The discussion about PT+ was intended to raise the issue of the need for local variables—but it also brings to light the problem of handling data structures that take multiple cells.

It can be a severe disadvantage to put bulky data structures of varying size directly on the parameter stack. Not only can it be hard to write the programs required to manipulate the data, but potentially it requires new sets of stack manipulation words. (If vectors occupy three cells and integers one cell, the user would presumably want words like VDUP, VSWAP, and mixed operations like VISWAP, etc. to allow Forth flexibility in implementing algorithms which involve the new data objects. This would become apparent as soon as anyone wants to use a word like PT+ to actually do something).

In some of the systems I've used for mathematics, there are six or more data types—each occupying a different number of cells. It has been found best to have every data object represented by a single cell on the stack (this representative is usually either an address, displacement within a segment set aside for the data type, or index in an array of objects). In this way, data objects can be subjected to all the usual Forth stack operations with no difficulty.

The data objects are equipped with words for accessing their component parts. Thus, for vectors we would want a word [] so that v 2 [] gives either the address of the second component of v or the second component itself. (If you choose the latter, you probably will also want a word []! to store data.) Chuck Moore seems to have originated the word TH for the component selector—so v 2 th. There will also be, presumably, words for arithmetic operations on the coordinates.

A vector package can be constructed in layers—so that it need not be just restricted to integer coordinates. Information about the coordinate domain is communicated to the level that handles vector operations—so the same code for vector operations can be used for a variety of different types of coordinates (here is another reason to make sure that different types of coordinates do not have different stack sizes).

The simplest way to do this (a better way is described below) is to have all operations specify the target of their result. Thus, for example, a vector addition V+ would have the stack diagram (v1 v2 v3 --) or maybe even (v1 v2 v3 -- v3) where the *vi* are "vectors" (i.e., the addresses of 3-tuples or the indices in an array of 3-tuples) and v3 is intended to be the result of adding v1 to v2. The code for V+ will be written in terms of an addition operation for components. (If preferred, you can write code for words like this with free use of ordinary Forth variables—no recursive calls are involved, so there is no need for local variables. Forth variables names can be reused—and they have a scope: they last until they are redefined. There is no harm in using the same variable names for independent procedures.)

This approach actually works, but it has the disadvantage that the syntax for operations is un-Forth-like. We would like V+ to have the stack diagram (v1 v2 -- v3) where v3 is not named—but just somehow appears. This can be accomplished by creating a set of temporary storage locations for each data type (I've been using 16 locations, and it seems to work just fine). Each time an operation is performed, the following happens:

1. The storage pool is searched for the next available address.
2. The operation is performed with the result stored at this address.
3. The address is put on the stack as the result of the operation—and it is marked as "unavailable" in the pool.

Once all addresses are marked, and no available address is found when requested, a mini-garbage collection takes place: the stack is examined to see which of the 16 addresses are still being referenced in the stack—all others are unmarked.

This reclamation is very fast—because only 16 addresses are involved and they are "used" only if found on the stack—so no big search through chains of pointers and variables is required. It is easily coded in assembly language. (The only caveat about this approach is that any data of lasting interest which is produced in one of these temporary locations should be moved to a permanent home. If a temporary address itself is stored in a variable, it will be considered available at the next garbage collection.)

I've used this approach for strings, several basic coefficient-type objects (like BIGNUM integers and BIGRAT rational numbers), compound objects (like polynomials with BIGNUM coefficients), etc. It works great as long as everything is represented on the stack using a single cell. It allows you to manipulate "gizmos" and "gadgets" without constantly being aware of what they look like, how big they are, and features of their internal representation. SWAP swaps a gizmo with a gadget, an integer with a string, etc.

(The code for this storage scheme for temporaries was written in Forth-83 using traditional Forth techniques. It takes about four screens. It was published in the *Journal of Forth Application and Research*—which seems to be defunct. I'm willing to mail people an electronic copy of the article if someone can assure me that this would not violate the *JFAR* copyright. I've never published in a journal that went out of business before, so I don't know the legalities involved. I also have fast versions for F83 and F-PC with selected words coded in assembly language.)

—John J Wavrik

From: Doug Philips

Jan Stout writes:

"Doug Philips writes:

"'Or perhaps

```
: (Pt+)  COMPILE R> COMPILE 3 COMPILE ROLL
COMPILE + ;
IMMEDIATE
```

"'Why go immediate?'"

I made it immediate so that the R> could be part of the + word.
—Doug

From: Doug Philips

John Wavrik writes:

"In some of the systems I've used for mathematics, there are six or more data types—each occupying a different number of cells. It has been found best to have every data object represented by a single cell on the stack (this representative is usually either an address, displacement within a segment set aside for the data type, or index in an array of objects). In this way data objects can be subjected to all the usual Forth stack operations with no difficulty."

I like that approach. I assume that you are not talking about having a typed stack though?

"Each time an operation is performed, the following happens:
"1. The storage pool is searched for the next available address.
"2. The operation is performed with the result stored at this address.
"3. The address is put on the stack as the result of the operation—and it is marked as 'unavailable' in the pool.
"Once all addresses are marked, and no available address is found when requested, a mini-garbage collection takes place: the stack is examined to see which of the 16 addresses are still being referenced in the stack—all others are unmarked."

I take it, then, that there is only one area of 16 items, regardless of the "type" of those items... or are you simplifying for the sake of discussion? And what happens if all 16 are in use? (I assume that can happen either because some other stack value "looks like" a reference to one of those cells, or because too many temporaries are—accidently?—being used.)

"I've used this approach for strings, several basic coefficient type objects (like BIGNUM integers and BIGRAT rational numbers), compound objects (like polynomials with BIGNUM coefficients), etc. It works great as long as everything is represented on the stack using a single cell. It allows you to manipulate "gizmos" and "gadgets" without constantly being aware of what they look like, how big they are, and features of their internal representation. SWAP swaps a gizmo with a gadget, an integer with a string, etc."

Indeed. But again, the programmer still has to know which cell on the stack is of what type (I'm not saying that is a bad thing...).

By the way, Upper Deck Forth uses a very similar scheme for handling strings. But instead of having explicit slots, usage flags, etc., their strategy is to use an area of memory as a ring buffer. Just keep a pointer to the next "free" address and a count of remaining bytes. If the new string object won't fit in the hole left at the end of the buffer, then wrap to the beginning. For a 1K buffer, that strategy guarantees at least four maximally sized strings simultaneously, and often a lot more. But the idea is nearly the same. Upper Deck has the advantage that there is no GC done whatsoever and it is totally up to the programmer to make sure that too many live strings are not "in use" at once.

Could you give a more complete reference to the *JFAR* article?
—Doug

From: John Wavrik

Concerning a data management scheme for temporaries, Doug Philips writes:

"I like that approach. I assume that you are not talking about having a typed stack though?"

The data stack is not typed, although it could be if all the addresses for a given address occupy one band of the address space. I generally like the Forth approach of producing differently named operations for each data type—rather than overloading an operator name and having run-time overhead based on typing of operands.

"'Each time an operation is performed, the following happens:
"'1. The storage pool is searched for the next available address.
"'2. The operation is performed with the result stored at this address.
"'3. The address is put on the stack as the result of the operation—and it is marked as 'unavailable' in the pool.
"'Once all addresses are marked, and no available address is found when requested, a mini-garbage collection takes place: the stack is examined to see which of the 16 addresses are still being referenced in the stack—all others are unmarked.'
"I take it then that there is only one area of 16 items, regardless of the 'type' of those items... or are you simplifying for the sake of discussion? And what happens if all 16 are in use? (I assume that that can happen either because some other stack value 'looks like' a reference to one of those cells, or because too many temporaries are—accidently?—being used)."

Each data type has its own set of 16 temporaries. The code that manages these is the same for all. A child of the defining word TEMP-STORAGE installs itself as the current data type. If STRINGS is one such data type, then STRINGS TEMP returns the address of the next temporary location for STRINGS. (The word STRINGS sets the current data type, and TEMP returns the next free address for the current data type.)

In general, all this is made invisible at the top level. The sequence

```
$" ABC"   $" XYZ"  OVER $+ $+
```

will put the (temporary) address of the string ABCXYZABC on top of the stack. Here, $" is state smart—but its interpret-time action is

```
: $"   STRINGS TEMP   ASCII " WORD   OVER $!  ;
```

where

```
: $!  OVER C@ 1+ CMOVE  ;
```

while $+ concatenates two strings, putting the result in a temporary and returning the address of the temporary to the stack.

If all 16 temporaries are in use, there is an ABORT. Except for programming errors, this has not occurred in practice. It is easy to make up examples where 16 locations could be insufficient (if you want to add 20 things, you could put them all on the stack and then apply addition 19 times—but you can also add as you go). It should be noted that H-P calculators have only four locations for temporaries—so 16 is probably too many.

"'I've used this approach for strings, several basic coefficient type objects (like BIGNUM integers and BIGRAT rational numbers), compound objects (like polynomials with BIGNUM coefficients), etc. It works great as long as everything is represented on the stack using a single cell. It allows you to manipulate 'gizmos' and 'gadgets' without constantly being aware of what they look like, how big they are, and features of their internal representation. SWAP swaps a gizmo with a gadget, an integer with a string, etc.'
"Indeed. But again, the programmer still has to know which cell on the stack is of what type (I'm not saying that that is a bad thing...)"

Yes—the programmer still has to know what is on the stack and its type—but not its size. No additional stack manipulation words need to be created. The storage management is concealed in the operation and input words, so becomes transparent for programming using these operations.

Example: for vectors there is an addition V+ and scalar multiplication cV* (applied with scalar on left).

```
a b V W  --  aV + bW
```

can be produced by

```
ROT SWAP cV*  -ROT cV*  V+
```

It doesn't matter which dimension the vector space is, because the components are not being stored on the stack. As a result, most of the code in a vector space package is independent of the size of vectors.

"By the way: Upper Deck Forth uses a very similar scheme for handling strings. But instead of having explicit slots, usage flags, etc., their strategy is to use an area of memory as a ring buffer. Just keep a pointer to the next 'free' address and a count of remaining bytes. If the new string object won't fit in the hole left at the end of the buffer, then wrap to the beginning. For a 1K buffer that strategy guarantees at least four maximally sized strings simultaneously, and often a lot more. But the idea is nearly the same. The Upper Deck has the advantage that there is no GC done whatsoever and it is totally up to the programmer to make sure that too many live strings are not 'in use' at once."

In this ring-buffer scheme, I assume that the addresses and lengths of strings still in use are tagged somehow, so that if the pointer circles the ring it will not allow the re-use of memory being used by an active string. Never having seen the Upper Deck system, I don't know how it determines if a string is still in active use.

"Garbage Collection" is actually a bit excessive a term for what is done. In my method it amounts to searching the stack (and perhaps somewhere else) to find which strings are no longer active—it is nowhere near what LISP must do to reclaim storage.

"Could you give a more complete reference to the *JFAR* article?"

"Handling Multiple Data Types in Forth" by John J. Wavrik. *JFAR*, vol. 6 no. 1 (1990).

—John J Wavrik

From: Doug Philips

John Wavrik writes:

"The data stack is not typed, although it could be if all the addresses for a given address *[you mean* type, *right?]* occupy one band of the address space. I generally like the Forth approach of producing differently named operations for each data type—rather than overloading an operator name and having run-time overhead based on typing of operands."

I agree. However, having a typed stack doesn't necessarily mean you have to use it to dispatch operators (though that is a pretty obvious thing to do with it). Even given the "address range"-typing kind of system you mentioned fails because it assumes that everything on the stack is a pointer.

"In this ring-buffer scheme, I assume that the addresses and lengths of strings still in use are tagged somehow, so that if the pointer circles the ring it will not allow the re-use of memory being used by an active string. Never having seen the Upper Deck system, I don't know how it determines if a string is still in active use."

There is no check. The premise is that the programmer knows how many maximally lengthed (counted) strings can be temporary at once, and so must not use any more than that. I cannot say that I am philosophically or practically bothered by their choice, but I probably would have done things differently.

My guess is that the biggest gotcha in either method is in keeping (or thinking that you can keep) temporary objects "live" over calls to non-trivial words.

"'Garbage Collection' is actually a bit excessive a term for what is done."

Indeed. Your method doesn't have the overwrite hole that the Upper Deck System's method does, but then neither method will catch a copy of the temporary pointer instead of copying the data.

—Doug

P.S. I elided all the stuff that I agreed with (mostly).

Contest Announcement

Forth Dimensions is sponsoring
a contest for articles about
"Forth on a Grand Scale"

Write about large-scale Forth applications, systems, or ...

"This theme applies equally to projects requiring multiple programmers, and to applications or systems consisting of large amounts of code and/or of significant complexity." ("Editorial," *FD* XIII/6) Papers will be refereed.

Mail a hard copy and a diskette (Macintosh 800K or PC preferred) to the:

Forth Interest Group
P.O. Box 2154 • Oakland, California 94621

Cash awards to authors:

1st place:	***$500***
2nd place:	***$250***
3rd place:	***$100***

Deadline for contest entries is August 3, 1992.

reSource Listings

Please send updates, corrections, additional listings, and suggestions to the Editor.

Forth Interest Group

The Forth Interest Group serves both expert and novice members with its network of chapters, *Forth Dimensions*, and conferences that regularly attract participants from around the world. For membership information, or to reserve advertising space, contact the administrative offices:

Forth Interest Group
P.O. Box 2154
Oakland, California 94621
510-89-FORTH
Fax: 510-535-1295

In Recognition

Recognition is offered annually to a person who has made an outstanding contribution in support of Forth and the Forth Interest Group. The individual is nominated and selected by previous recipients of the "FIGGY." Each receives an engraved award, and is named on a plaque in the administrative offices.

1979 William Ragsdale
1980 Kim Harris
1981 Dave Kilbridge
1982 Roy Martens
1983 John D. Hall
1984 Robert Reiling
1985 Thea Martin
1986 C.H. Ting
1987 Marlin Ouverson
1988 Dennis Ruffer
1989 Jan Shepherd
1990 Gary Smith
1991 Mike Elola

ANS Forth

The following members of the ANS X3J14 Forth Standard Committee are available to personally carry your proposals and concerns to the committee. Please feel free to call or write to them directly:

Gary Betts
Unisyn
301 Main, penthouse #2
Longmont, CO 80501
303-924-9193

Charles Keane
Performance Pkgs., Inc.
515 Fourth Avenue
Watervleit, NY 12189-3703
518-274-4774

Mike Nemeth
CSC
10025 Locust St.
Glenndale, MD 20769
301-286-8313

George Shaw
Shaw Laboratories
P.O. Box 3471
Hayward, CA 94540-3471
415-276-5953

Andrew Kobziar
NCR
Medical Systems Group
950 Danby Rd.
Ithaca, NY 14850
607-273-5310

David C. Petty
Digitel
125 Cambridge Park Dr.
Cambridge, MA 02140-2311

Elizabeth D. Rather
FORTH, Inc.
111 N. Sepulveda Blvd.,
suite 300
Manhattan Beach, CA 90266
213-372-8493

Forth Instruction

Los Angeles—Introductory and intermediate three-day intensive courses in Forth programming are offered monthly by Laboratory Microsystems. These hands-on courses are designed for engineers and programmers who need to become proficient in Forth in the least amount of time. Telephone 213-306-7412.

(Letters, from page 5.)

I/O Time Dominates Real Performance

Dear Marlin:

I heartily agree with Jim Callahan's remarks concerning the inadequacy of relying solely on speed benchmarks when evaluating Forth implementations. Not only is it important to look at the full set of tradeoffs (such as compactness, etc.) as he points out, it is also important to remember that the aspects of performance measured by this set of benchmarks don't give an accurate picture of the overall performance of an application. In most real-world applications, performance is overwhelmingly dominated by the time required for I/O. A system that offers low-overhead interrupt handling and high-speed multitasking such as polyFORTH can often out-perform other run-time environments.

However, Callahan's remarks about polyFORTH are inaccurate. The multi-segment polyFORTH uses, in its minimum configuration, one code area up to 64K (for machine code, definition pointers, and heads) and one data area up to 64K (for disk buffers, global variables, and task space, including stacks and user variables). Between these areas there is no duplication. Extra memory (up to DOS' 640K limit) may also be configured. Typically, such extra memory is used for large data structures, but it may also be configured for additional code modules. In the latter case only, routines used by those modules are replicated. The amount of replicated code is controlled by the programmer, but rarely exceeds about 21K. The cost of avoiding this replication would be a substantial speed penalty imposed on routines in the extra code modules, as well as extra bytes required for addresses. We have found it to be an appropriate tradeoff.

In today's market, substantial PC applications are generally written for a 386/486. As noted in Kelly's article, we also offer a 32-bit, protected-mode system which is well suited for large applications, leaving the 16-bit, segmented model as an economical solution for low-end applications.

Sincerely,
Elizabeth D. Rather, President
Forth, Inc.
111 N. Sepulveda Blvd.
Manhattan Beach, California 90266-6847

CREATE ... DOES> Erratum

Dear Marlin. Ijust received the new issue of *FD*[XIV/1]. I am glad you are are recruiting more tutorials. I have found an error in my article. In Screen 3, `MAKE-8` reads,

```
: MAKE-8 ( i -- a) SWAP 8 * + ;
```

It should read

```
: MAKE-8 ( i -- a) DOES> SWAP 8 * + ;
```

—Leonard Morgenstern

RIME (Relay International Message Exch.) Forth Conf. Echo

RIME is a PC-Board-based network similar to FIDO. The Forth Conference originates on Jim Wenzel's Grapevine BBS in Little Rock, Arkansas. Messages carried on RIME Forth Conference are essentially identical to those carried on the GEnie Forth RoundTable and on UseNet comp.lang.forth newsgroup.

BBS Name	**City**	**Phone**
Evergreen BBS	Hopatcong, NJ	201-398-2373
Bob'S Corner Board	Gainesville, FL	205-361-9094
Country Lane	Kennenbunk, ME	207-499-2756
The Running Board	Bronx, NY	212-654-1349
AmerIServe	New York, NY	212-876-5885
Ground Zero Wildcat BBS	Seal Beach, CA	213-430-0079
The Holistic BBS	Lakewood, CA	213-531-3890
Shy Guy's PCBoard	Lewisville, TX	214-315-3795
DFW Programmer's Exchange	Dallas, TX	214-398-3112
The Lunatic Fringe BBS	Piano, TX	214-422-2936
Ronin	Waxahachie, TX	214-938-2840
The Round Table BBS	Reading, PA	215-678-0818
Street Noise! BBS	Germantown, MD	301-601-8710
Baudline II	Frederick, MD	301-694-7108
Network East	Rockville, MD	301-738-0000
The Jellicle Cate	Riverdale, MD	301-779-5946
The Carousel	Hollywood, FL	305-987-5688
Back to Basics	Casper, WY	307-235-7043
Travel Online	Lake St. Louis, MO	314-625-4045
M.O.R.E.	Middletown, RI	401-849-1874
Castle Rock BBS	Omaha, NE	402-572-8247
D.W.'s Toolbox	Riverdale, GA	404-471-6636
The Right Place tm)	Atlanta, GA	404-476-2607
The Chair TOO!	San Jose, CA	408-241-7276
PDS-SIG BBS	San Jose, CA	408-270-4085
The Caves	Scotts Valley, CA	408-438-1194
Eds Home	Columbia, MD	410-730-2917
ProPC BBS	Pittsburgh, PA	412-321-6645
PGHSouth PCBoard System	Pittsburgh, PA	412-563-5416
Space BBS	Menlo Park, CA	415-323-4193
Mental Hospital	Los Altos, CA	415-941-5384
Canada Remote Systems	Misssissauga, CN	416-629-0136
Rose Media	Willowdale, CN	416-733-2285
The Grapevine BBS	N. Little Rock, AR	501-753-8121
The GrapeVine Remote Node III	Little Rock, AR	501-821-4827
The Pegasus BBS	Owensboro, KY	502-684-9855
Alpine BBS	Salem, OR	503-581-0923
The Crooked Blade	Monmouth, OR	503-838-4059
River Road BBS	Sorrento, LA	504-675-8792
IDC BBS	Alameda, CA	510-865-7115
Modem Zone	Middletown, OH	513-424-7529
Channel 1	Cambridge, MA	617-354-8873
Capital Connection	Fairfax, VA	703-280-5490
Hallucination BBS	Fairfax, VA	703-425-5824
No-Frills BBS	Falls Church, VA	703-538-4634
Struppi's BBS	Herndon, VA	703-620-2646
The Virginia Connection	Reston, VA	703-648-1841
The Beltway Bandits BBS	Fairfax, VA	703-764-9297
Programmer's Palace	Springfield, VA	703-866-4452
Technet At TJHSST	Alexandria, VA	703-941-3572
Carolina Forum	Charlotte, NC	704-563-5857
Nezuld's Domain	Northbrook, IL	708-559-0513
Aquila BBS	Aurora, IL	708-820-8344
Cloud Nine BBS	Katy, TX	713-859-8195
The Punkin Duster BBS	Fullerton, CA	714-522-3980
Crystal Castle	Staten Island , NY	718-370-8031
Moondog	Brooklyn, NY	718-692-2498
The Icebox BBS	Flushing, NY	718-793-8548
Rocky Mountain Software	Salt Lake City, UT	801-963-8721
Club PC BBS	Smithfield, VA	804-357-0357
The Computer Forum BBS	Virginia Beach, VA	804-471-3360
The Godfather	Tampa, FL	813-289-3314
St. Pete Programmer's Exchange	St Petersburg, FL	813-527-5666
DataBoard][BBS	Crowley, TX	817-297-6222
The Mog-Ur's EMS	Granada Hills, CA	818-366-1238
Medical Information Systems	Jacksonville, FL	904-221-9425
The Enchanted Forest BBS	Gainesville, FL	904-377-2001
The TREE BBS	Ocala, FL	904-732-0866
Ramwood	Fairbanks, AK	907-456-6375
The Imperium BBS	Middletown, NJ	908-706-0213
PC Rockland BBS	South Nyack, NY	914-353-2157
The Pub BBS	White Plains, NY	914-686-8091
Brentwood BBS	Harrison, NY	914-835-1315
Technical Information	Taipei, TW	01188622151127
O.L.E.F.1	London, UK	44-81-882-9808

On the Back Burner

Sleeping with the Enemy

Conducted by Russell L. Harris
Houston, Texas

The inaugural Back Burner column broached the problem of obtaining authorization of a client or boss to use Forth on a particular contract. In that column, I proposed demonstration apparatus as a possible approach to securing such authorization. In this column, risking a fusillade of brickbats and charges of heresy, I wish to suggest another possibility, that of utilizing a mixture of Forth and C (or whatever the language to which your firm or client is committed).

The general idea is to get your foot in the door by agreeing to do the project in the language specified. Being knowledgeable in Forth, you will inevitably utilize Forth to facilitate development and testing. This code comprises your primary sales tool. As such, it should be written and commented with exceptional care. What you are counting on is the intelligence, rationality, and objectivity of the boss or client. Once you can show him functional Forth code side by side with functional code written in C, etc, in the same project, it should not be difficult to make your case. If the ploy doesn't work, at least you have employment and another C project under your belt.

The inconsistency of C is simply taken for granted and, at times, even praised as versatility.

The Carrot on the Stick

Granted, learning C in order to get projects on which you hope to use Forth may not appear to be a sound proposition. However, there is another incentive: C is rapidly becoming the *lingua franca* in which program algorithms are presented. This is particularly true in the realm of digital signal processing algorithms. Revolting as the thought may be, familiarity with the rudiments of C may soon become as vital as familiarity with the rudiments of MS-DOS.

Flimflam, or Malice in Blunderland

C is a complex language which is difficult to master. Complexity itself is not the culprit; rather, the difficulty springs from inconsistency.

C is a language of innumerable rules. Moreover, there are far more exceptions than there are rules: everything seems to be a special case. When programming in C, one cannot rely upon logic or intuition to guide him in syntactical construction or in deducing the behavior of a code segment. One simply must learn C by rote.

Like something from the pen of Lewis Carroll, C resembles a child's game in which the participants make the rules as they go along, whimsically changing them in order to thwart one another's progress. The strange thing is that no one (other than programmers knowledgeable in assembler or Forth) seems to think the situation strange, improper, unnecessary, or inexcusable. The inconsistency of C is simply taken for granted and, at times, even praised as versatility. Indeed, reading Kernighan & Richie's *The C Programming Language* brought to my mind the old adage of product advertising that goes something like this: "If you can't fix it or hide it [referring to a glaring deficiency], tout it as a feature."

A Plan of Attack

Needing to learn the rudiments of C, I acquired, upon the advice of a colleague, a copy of the second edition of Voss & Chui's *Turbo C++ Disktutor.* Before you rush out and buy a copy, let me warn you that the book contains an irresponsible number of formatting and typographical errors, particularly for a second edition, and is in dire need of the services of a competent editor.

Nevertheless, I found Voss & Chui a useful framework in which to approach the formidable world of C and object-oriented programming. The authors take the reader step by step through construction of a windowing system for the IBM-PC/MS-DOS environment, definitely a useful goal.

What's in a Name?

As astute readers by now have discerned, there is a correlation between the name of this column and the placement thereof within the pages of this publication. The inspiration I owe to Ed Zern, author of the "Exit, Laughing" column which occupies the corresponding location in the outdoors magazine *Field & Stream* (not to be mistaken for the horror magazine, published in Braille and bearing the title *Feel & Scream*).

There is, however, a more serious side to selection of the title, and this is in keeping with the kitchen stove analogy. In most commercial endeavors, the back burner is where in-house advances are made, whereas activity on the front burners meets the payroll and pays the bills. Without research-and-development projects on the back burner, advances in capability are limited to tools and techniques developed by others and made available on the commercial market or described in the literature.

It is my intention that this column serve as inspiration for back-burner development. Reader interaction is essential, if the column is to be successful and continue. R.S.V.P.

Russell Harris is a consulting engineer working with embedded systems in the fields of instrumentation and machine control. He can be reached by phone at 713-461-1618, or at his RUSSELL.H address on GEnie.

NEW FROM THE FORTH INTEREST GROUP

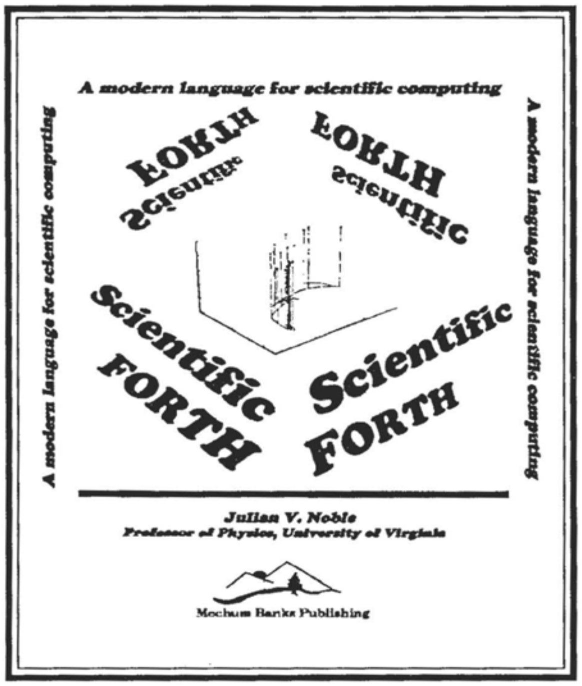
"...FORTH is not usually encountered within the context of scientific or engineering computation, although most users of personal computers or workstations have unwittingly experienced it in one form or another. FORTH has been called 'one of the best-kept secrets in computing'. It lurks unseen in automatic bank teller machines, computer games, industrial control devices and robots...

"Some scientists and engineers have gained familiarity with FORTH because it is fast, compact, and easily debugged; and because it simplifies interfacing microprocessors with machines and laboratory equipment...

"...FORTH has the ability not only to reproduce all the functionality of FORTRAN—using less memory, compiling much faster and often executing faster also—but to do things that FORTRAN could not accomplish easily or even at all...

"One reason FORTH has not yet realized its potential in scientific computing is that scientists and programmers tend to reside in orthogonal communities, so that no one has until now troubled to write the necessary extensions. One aim of this book is to provide such extensions in a form I hope will prove appealing to current FORTRAN users.

"Since time and chance happen to everything, even FORTH, I have devoted considerable effort to explaining the algorithms and ideas behind these extensions, as well as their nuts and bolts..."



A modern language for scientific computing

Scientific FORTH

Julian V. Noble
Professor of Physics, University of Virginia

Mechum Banks Publishing

Scientific Forth
by Julian V. Noble

Scientific Forth extends the Forth kernel in the direction of scientific problem solving. It illustrates advanced Forth programming techniques with non-trivial 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. (Includes disk with programs and several utilities.)

$50.00

Forth Interest Group
P.O. Box 2154
Oakland, CA 94621

Second Class
Postage Paid at
San Jose, CA