$10 Volume XIV, Number 3 September 1992 October

FORTH DIMENSIONS

A Little Help Engine

Principles of Metacompilation

Structured Pattern Matching (II)

What's in a Standard?

Contents

Features

6 The Little Help Engine
Frank Sergeant

On-line help is *de rigeur* for many applications today, and even for some language implementations. Enhancing your programs in this way makes them easier to use, adds to their apparent completeness and value, and demonstrates the programmer's awareness of user considerations. The author shares such a utility in his signature style: small, simple, and fully functional.

13 Principles of Metacompilation, *Part One*
B.J. Rodriguez

Metacompiler mystique is compounded by the dearth of "what, why, and how" documentation for this powerful feature of Forth. This well-known Forth expert dispells the difficulties that have prevented many Forth users from mastering the art and science of metacompilation. With such knowledge, you can experiment with re-tooled versions of your Forth, create a new Forth for a different processor, and optimize embedded applications.

22 Structured Pattern Matching
Ariel Scolnicov

The code presented here implements the ground-breaking work published in our last issue. With it, you can define patterns to describe classes of strings, then use the search engine to find permutations of pattern combinations. A Forth user at Bell Labs writes that these definitions "...go a long way to implement Unix System Tools functionality" and form the basis of Forth replacements for *grep*, *sed*, and *awk*.

Departments

4 Editorial ..... Endangered Species?

4 FIG Financial Statement

5 Letters ..... Paging through RAM with CREATE ... DOES>, Building on Monumental Analysis, the Future of Forth and FIG, Pattern Matching Rings True.

21 Advertisers Index

30 Fast Forthward ..... The Changing Marketplace; plus Forth press coverage and new products.

32 Best of GEnie ..... On-line debate over what ANS Forth should include; different ways to make a standard; impact of a standard on extant code.

37–41 reSource Listings ..... Forth Interest Group, ANS Forth, classes, on-line resources, FIG chapters.

43 On the Back Burner ... The Fireman Syndrome: going where no others dare.

Editorial

Endangered Species?

Salmon swimming upstream is an overworked analogy, but it may apply well enough to those who insist that large and complex Forth systems contradict an essential virtue of Forth. Several items in this issue remind us that, regardless of personal and philosophical preferences, most programmers get employment today by working in large systems. Those who won't adapt their Forthly ways to accommodate the demands of such environments don't get that kind of work, or must abandon Forth along with *all* its virtues.

Has anyone satisfactorily answered the question, "What is Forth?" Is it smallness and permutability, or is it also elegance and a way of thinking about software tools and access to the fundamental hardware? Can we apply its principles universally, to feel equally potent in multimeg memory and applications? Can we (we can, but should we?) communicate with the rest of the computing world, or must we focus on standalone (e.g., embedded) systems? Some say that small applications that require no OS and little user interface also comprise a smaller (and less visible) market, one easily saturated both by products and by programmers. They say, adapt Forth or die! And maybe, just a little, they want to prove that Forth really could be a contender...

Still, those contrary and willful salmon—just barely enough of them—do make their way through opposing currents, negotiating bears, rapids, and man-made obstacles. They still die, in the end, but at least they get to the spawning before their life cycle ends.

Can two different conceptualizations of Forth coexist under the same name, or must one be right and the other doomed? Standards, a FORML committee once declared, are best viewed as tools for communication; in practice, Forth has many faces, and each is beautiful in the eyes of the implementor. As always, Forth will be what the Forth community defines/allows it to be.

Maybe the question should not be "What is Forth" but "Where is Forth at home?" That shifts the emphasis from an ideological to a practical note, while challenging Forth ideologues to test their convictions under contemporary conditions. Practical tools, after all, will always provide more income to more programmers. And, like spawning salmon, the more the merrier and the more likely it is that the species as a whole will prosper.

Happily, while arguments about survival of the fittest continue, we have a wide selection of Forth implementations, and they represent a large spectrum of ideologies. Some are optimized for tiny embedded applications, others are well-adapted to Macintosh, Amiga, and OS/2 environments, among many others. Some are of the lean, mean, roll-it-yourself variety, while others are so feature-rich as to entice even the most spoiled programmers.

Regardless of your own programming proclivities, we'd like to hear from you. *FD* needs articles and letters from its readers, who are always our main inspiration. Tell us how you have used, extended, or modified Forth: share what you've learned!

Forth Interest Group
Statement of Change in Financial Position
Apr 30, 1991 to Apr 30, 1992

ASSETS:	4/30/91	4/30/92	Change + = Increase - = Decrease
Current Assets:			
Money Market	41,782.37	33,956.22	-7,826.15
Checking	1,936.78	2,845.94	909.16
Pending Foreign Clearing	51.67	51.67	0.00
Returned Checks Pending	72.00	110.00	38.00
Total Current Assets:	43,842.82	36,963.83	-6,878.99
Inventory:			
Inventory at cost	26,601.17	24,600.57	-2,000.60
Total Inventory:	26,601.17	24,600.57	-2,000.60
Other Assets:			
Deposit, United Parcel Service	200.00	200.00	0.00
Second Class Postal Account	192.41	174.51	-17.90
Accounts Receivable	2,099.00	1,285.50	-813.50
Total Other Assets:	2,491.41	1,660.01	-831.40
TOTAL ASSETS:	72,935.40	63,224.41	-9,710.99
LIABILITIES:			
Sales Tax	35.58	46.66	11.08
FD Dues Alloc to future months	41,518.51	30,289.20	-11,229.31
TOTAL LIABILITIES:	41,554.09	30,335.86	-11,218.23
Financial Reserve:	31,381.31	32,888.55	1,507.24

Forth Dimensions
Volume XIV, Number 3
September 1992 October

Published by the
Forth Interest Group

Editor
Marlin Ouverson

Circulation/Order Desk
Frank Hall

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

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

The Forth Interest Group

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

"*Forth Dimensions* (ISSN 0884-0822) is published bimonthly for $40/46/52 per year by the Forth Interest Group, 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!

Paging through RAM with CREATE ... DOES>

Dear Marlin,

I enjoyed Leonard Morgenstern's article on CREATE ... DOES> (*FD* XIV/1). One aspect of the subject that might be worth a mention is the ability of a defining word to choose from a selection of DOES> or ;CODE routines for the words it defines. For example, many constants are small positive numbers that can be contained in a single byte. This can be exploited to economize on space:

```
: CONSTANT ( n -- ) ( -- n )
  CREATE DUP 256 U<
  IF
    C, DOES> C@ EXIT
  THEN
  , DOES> @ ;
```

Or in assembler (the following is in F83, but ;CODE must be redefined, leaving out the compile-time stack depth check):

Forth is the better programming philosophy; C has more subroutine libraries. Both together would be the optimum.

```
: CONSTANT ( n -- ) ( -- n )
  CREATE DUP 256 U<
  IF
    C, ;CODE
    2 [W] AL MOV
    0 # AH MOV
    AX PUSH
    NEXT FORTH ]
  THEN
  , ;CODE
  2 [W] PUSH
  NEXT  END-CODE
```

In theory, any constant could be accommodated in a single byte by storing only the low-order byte with the defined word and having the high-order byte supplied by the appropriate DOES> or ;CODE routine. On some processors, this can be done in assembler without incurring any execution-speed penalty. In practice, there are seldom enough constants in any particular range (except for 0 to 255) to make the space overhead of the extra code worthwhile.

An exception to this occurs in the case of ROMable variables, whose data fields are in an area of memory separate from the main dictionary. At run time, such a variable is effectively a constant whose value is the address of the associated data field. A simple high-level definition might be:

```
: VARIABLE ( -- ) ( -- addr )
  CREATE HERE(R)
  , 2 ALLOT(R)
  DOES> @ ;
```

where HERE(R) and ALLOT(R) play the same role in relation to the data area of memory that HERE and ALLOT perform with respect to the main dictionary ("R" stands for RAM—cleverly distinguishing it from ROM). Since many variables may have data fields on the same 256-byte page of RAM, it can make sense to provide a separate DOES> or ;CODE routine for each page used. When I have employed this technique in the past, I have used explicit RAM addresses in the definition of VARIABLE, but Morgenstern's NUMBER-MACHINE has inspired a more generalized form:

```
HEX
: HALF-PAGER
  ( addr -- ) ( -- page-aligned-addr )
  CREATE 0FF00 AND , IMMEDIATE
  DOES> DUP @ 80 ROT +! 0FF00 AND
  STATE @ IF [COMPILE] LITERAL THEN ;
```

A word defined by HALF-PAGER always returns a page-aligned address, but it takes two references to flip it over to the next page. It is state smart, so it can be used in high-level Forth or assembler.

```
HERE(R) HALF-PAGER RAMPAGE
: VARIABLE ( -- ) ( -- addr )
  CREATE HERE(R) 2 ALLOT(R) DUP
    0FF00 AND
  DUP RAMPAGE =
    IF DROP C, DOES> C@ RAMPAGE
    OR EXIT THEN
  DUP RAMPAGE =
    IF DROP C, DOES> C@ RAMPAGE
    OR EXIT THEN
  DUP RAMPAGE =
    IF DROP C, DOES> C@ RAMPAGE
    OR EXIT THEN
    ( ...repeat for however many pages of RAM... )
  DROP , DOES> @ ;
```

(Continued on page 10.)

The Little Help Engine

Frank Sergeant
San Marcos, Texas

I wanted on-line Help for my latest project, the Bare Bones EPROM Programmer kit. The clearer the documentation, the happier the users and the easier the technical support. Here is the code in Pygmy Forth for the Help system, along with a discussion of various alternatives and enhancements. You can use this approach with just about any Forth system to add Help to your own applications.

What should happen when F1 is pressed? Pop up the message "Read The Manual?" No, we need a richer facility, but without spending too much time or memory on it.

The Basics

Later we will add some features, but first let's consider a simple Help system. Pressing PgDn or PgUp moves through the entire Help system sequentially. Pressing a number key (i.e., 0 to 9) jumps directly to a selected topic. Pressing Esc (escape) exits the Help system. It must be easy for the developer to add and change the text and the links between topics. Changing the text does not require re-building or re-compiling the application. Thus, the Help "engine" is independent of the specific text and application.

What should happen when F1 is pressed? Pop up the message "Read The Manual?" No, we need a richer facility...

The Help system is invoked from application code by specifying a range of Forth blocks, as in:

```
3000 3031 HELP
```

HELP then clears the screen and displays the first 15 lines of the first block (block #3000 in the example above). The next block to be displayed depends on the key the user presses. As mentioned above, PgUp and PgDn move sequentially through the allowable range of blocks, and Esc ends the Help session. The last line of the block contains the block numbers to jump to if the user presses the keys 0 to 9. These destinations are in plain ASCII, five digits per number, with a single space between numbers. If any other key is pressed, HELP beeps and ignores it. Since both the text and the links are in plain ASCII format, no special compilation step is needed. Type the text and any destination numbers with your regular block editor. To allow the user to jump from the current block, give him a numbered list of choices and put the associated destinations on the bottom line of the block.

The entire Help engine consists of just five or six Forth words and takes very little space in the dictionary, no matter how simple or fancy the Help text may be. See Figure One for the source code.

NUM (a # -- n) is a replacement for NUMBER (a -- n). Your Forth system may already have such a word. If so, use it instead. Unfortunately, my NUMBER takes a counted string (and actually uses the count), and I do not wish to embed a count into the Help blocks. Later I will probably change my NUMBER to take an address and count. Meanwhile I use NUM for this purpose.

The variable HELP# (-- a) holds the number of the current Help block.

INDEX@ (choice -- next-help-scr#) uses the choice number of 0 to 9 to index into the list of numbers on the bottom line of the current Help block. Each number occupies five characters and is followed by a space. For example, the number associated with choice three starts 18 bytes from the beginning of the last line. The word -LEADING eats leading spaces, so we can omit leading zeroes. We must still right-justify the numbers in their proper five-byte fields. Alternatively, we can use -TRAILING to eat trailing spaces, and left-justify the numbers. Unused choices may be left blank.

.HELP (--) clears the screen and displays the first 15 lines of the current Help block.

Based on the key the user presses, NEXT-HELP# (key -- new-scr#) proposes a number for the next Help block to be displayed.

Figure One. All the code necessary for a simple Help Engine.

```
file HELP.SCR

scr # 2000
|EXIT                  Help System Summary                          |
|     Display the first 15 lines of the current help screen.        |
|     Wait for a keypress:                                          |
|          Esc ends the help session.                               |
|          PgUp or PgDn moves sequentially through help screens.    |
|          0-9 selects a new help screen to jump to.                |
|     When jumping to a screen, the key 0-9 is used as an index     |
|into the last line of the current help screen to find the number|
|of the next help screen.  See bottom of this screen for an         |
|example, where pressing key 0 jumps to screen 12345 and pressing|
|key 1 jumps to screen 5.  Right justify numbers in a 5-character|
|field, follow each number by a space.  Leading zeroes are          |
|optional.                                                          |
|                                                                   |
|Example:                                                           |
|12345 00005 00000   345  2341    75    76    77 64432   119      |
|___________________________________________________________________|

scr # 2001
|( Help                NUM    HELP#)                                |
|                                                                   |
|        ( replacement for NUMBER that uses an address and count)|
|: NUM ( a # - n)                                                   |
|  DUP PAD C!     PAD 1+ SWAP CMOVE (  )  PAD NUMBER   ;            |
|                                                                   |
|                                                                   |
|VARIABLE HELP#              ( block # of current help screen)      |
|                                                                   |
|                                                                   |
|                                                                   |
|                                                                   |
|___________________________________________________________________|

scr # 2002
|( Help                INDEX@)                                      |
|                                                                   |
|( Choose next help screen from list at bottom of current screen)|
|                                                                   |
|: INDEX@ ( choice - next-help-scr#)                                |
|  6 *                            ( ie 6 bytes per choice number)   |
|  HELP# @ BLOCK   ( ie starting address of current help screen)    |
|  [ 15 64 * ] LITERAL +                ( ie start of last line)    |
|  +                                 ( addr-of-selected-number)     |
|  5 -LEADING ( a #)                ( ie eat any leading spaces)    |
|  NUM  ;                                                           |
|                                                                   |
|                                                                   |
|                                                                   |
|___________________________________________________________________|

scr # 2003
|( Help                .HELP)                                       |
|                                                                   |
|                ( show the 1st 15 lines of current help screen)    |
|                                                                   |
|: .HELP ( # -)                                                     |
|  CLS                                              ( clear screen) |
|  HELP# @ BLOCK ( a)                                               |
|  15 FOR ( a)                                                      |
|         DUP 64 -TRAILING TYPE  CR                  ( type a line) |
|         64 +                                ( advance to next line)|
|     NEXT  DROP  ;                                                 |
|                                                                   |
|                                                                   |
|                                                                   |
|___________________________________________________________________|
```

HELP (1st-scr# last-scr# --) puts it all together, displaying the first block, collecting keystrokes, and moving to the next block.

I've spread the code out with extra lines and screens and comments to make it more readable, but you can probably crowd it onto a single screen if you wish.

Alternatives

The simple system described above is easy to implement, surprisingly powerful, and takes about 336 bytes of dictionary space. However, it has some limitations. First, the text displayed is limited to 15 lines of 64 characters. This can be fancied up by surrounding the text with a box, the title "HELP," and a message at the bottom suggesting which keys could be pressed next. See the alternate definition of .HELP in Figure Two.

Perhaps 15 lines of 64 characters are enough. It forces you to be concise when you create the text, or to spread it out in bite-sized pieces among several screens. This may well improve the ease of use and the clarity of the system from the user's viewpoint.

Another limitation is the ten direct-jump destinations, one for each of the digits 0 to 9. A list of 15 or 20 destinations must be broken down into groups and subgroups so that, at each level, you have no more than ten choices. Since humans are usually more comfortable with a list beginning with one than with zero, you may wish to further limit the number of choices to only nine.

If we need more than nine or ten choices, we can allow letters and/or larger numbers. This requires several key-

strokes from the user and a more complex Help engine. The single keystroke is easier on the user, I think, and on the programmer. Various other approaches, such as making certain words appear in bold, and tabbing to them, could be tried, but I think they cost too much in complexity for what they give. All in all, pressing a key between 0 and 9 is pretty convenient.

The limitation of 15 lines by 64 characters can be gotten around, and may be worth doing at times. For example, you might set up the Help blocks in pairs, where 16 lines of the first and eight lines of the second are displayed. Then you have lots of room remaining on the second block to associate letter or digit keypresses with destinations. Some combination of these variations should handle just about any Help system need.

Backing Up

This is a hierarchical, direct-access system, not purely sequential. As such, it might be nice to be able to backtrack. Three possibilities come to mind. The first is to stack each screen as we move to the next. Then, whenever the we press Esc, we back up to the previous screen, until we are at the root. Then Esc exits the Help system. This is no good: If we traverse the entire system sequentially, we surely should not need to revisit every single screen again just to get out of Help.

A more reasonable approach is to stack only the screens we jump from. When we move sequentially with PgUp and PgDn, nothing gets stacked. As before, Esc backs up, but to the last place we *jumped* from. This could be

```
scr # 2004
( Help                  NEXT-HELP#)
                                ( propose a new help screen number)
: NEXT-HELP# ( key - scr#)
   HELP# @ SWAP  ( old-scr# key)
   DUP 201 ( PgUp) = IF DROP 1- EXIT THEN        ( ie increment)
   DUP 209 ( PgDn) = IF DROP 1+ EXIT THEN        ( ie decrement)
   DUP '0 '9 BETWEEN IF
            '0 -    ( ie >DIGIT, convert character to a number)
            INDEX@                ( and then to a block number)
            SWAP DROP EXIT THEN
   DROP ( old-scr#) ;         ( for illegal key return old-scr#)

scr # 2005
( Help                      HELP)
: HELP ( 1st last -)
  OVER HELP# !   ( make the 1st screen the current help screen)
  BEGIN ( 1st last)
      .HELP                  ( display the current help screen)
      KEY DUP 27 -
    WHILE ( while not Esc)
      NEXT-HELP#  ( convert the key to a possible next screen)
      ( 1st last new) PUSH 2DUP ( 1st last 1st last)
      R@ ROT ROT BETWEEN           ( ie is new scr# in-range?)
      R@ HELP# @ -  ( ie is new scr# different from current#?)
      AND ( 1st last flag)   ( must be in-range and different)
      POP SWAP ( 1st last scr# flag)
      IF HELP# ! ELSE DROP BEEP THEN  ( 1st last)
  REPEAT ( 1st last key) DROP 2DROP  ;
```

Figure Two. Alternate definitions for a fancier Help engine.

```
scr # 2006
( Help    Alternate version of .HELP)
                ( show the 1st 15 lines of current help screen)
: .HELP ( # -)
  CLS                                          ( clear screen)
  ." ---------------------------- Help -------------------------"
-----" CR   64 SPACES ." |" CR
  HELP# @ BLOCK ( a)
  15 FOR ( a)
         DUP 64 TYPE ." |" CR          ( type a line)
         64 +                           ( advance to next line)
    NEXT  DROP  64 SPACES ." |" CR
  ." ------------------------------------ Esc--PgUp--PgDn--0 th
ru 9 "  ;

scr # 2007
( Help       Alternate version of NEXT-HELP#)
                            ( propose a new help screen number)
: NEXT-HELP# ( key - scr#)
   HELP# @ SWAP  ( old-scr# key)
   DUP 201 ( PgUp) = IF DROP 1- EXIT THEN        ( ie increment)
   DUP 209 ( PgDn) = IF DROP 1+ EXIT THEN        ( ie decrement)
   DUP 27 ( Esc) = IF 2DROP 0 INDEX@ EXIT THEN  ( ie backtrack)
   DUP '0 '9 BETWEEN IF
            '0 -    ( ie >DIGIT, convert character to a number)
            INDEX@                ( and then to a block number)
            DUP IF SWAP THEN DROP   ( retain old-scr# if new=0)
            EXIT THEN
   DROP ;
```

```
scr # 2008
|( Help  Alternate version of HELP, Esc backs up through tree)    |
|: HELP  ( 1st last -)                                            |
|  OVER HELP# !    ( make the 1st screen the current help screen) |
|  BEGIN  ( 1st last)                                             |
|       .HELP                    ( display the current help screen) |
|        KEY                                                      |
|        NEXT-HELP#   ( convert the key to a possible next screen) |
|  ( 1st last new) ?DUP WHILE          ( while screen# is not zero) |
|        PUSH 2DUP  ( 1st last 1st last)                          |
|        R@ ROT ROT BETWEEN              ( ie is new scr# in-range?) |
|        R@ HELP# @ -   ( ie is new scr# different from current#?) |
|        AND  ( 1st last flag)    ( must be in-range and different) |
|        POP SWAP  ( 1st last scr# flag)                          |
|        IF HELP# ! ELSE DROP BEEP THEN   ( 1st last)             |
|  REPEAT  ( 1st last)   2DROP   ;                                |
|                                                                 |
|_________________________________________________________________|
```

Figure Three. Excerpts from the Bare Bones EPROM Programmer Help system, showing the links on the bottom line of each screen.

```
file EPROM.HLP
scr # 3000
|                                        How to Use the Help System|
|                                                                  |
|     Use the PgUp and PgDn keys on the numeric keypad to browse   |
|sequentially through the help screens.                            |
|                                                                  |
|     Some screens have a list of numbered topics.  To jump        |
|directly to a topic, press its corresponding number.              |
|                                                                  |
|     Press Esc to get out of HELP and return to the main menu.    |
|                                                                  |
|               1. Overview.                                       |
|               2. Copyright Notice.                               |
|               3. Index of topics.                                |
|               4. Definitions.                                    |
|               5. How to Program an EPROM.                        |
|00000 03003 03001 03002 03034 03005                               |
|__________________________________________________________________|

scr # 3001
|                                                  Copyright Notice|
|                 The Bare Bones EPROM Programmer                  |
|                                                                  |
|     Programs, documentation, and printed circuit board           |
|copyright (c) 1992 by Frank Sergeant                              |
|                       809 W. San Antonio Street                  |
|                       San Marcos, Texas  78666                   |
|                                                                  |
|     Press Esc to get out of HELP.                                |
|               1. Index of topics.                               |
|               2. Overview.                                       |
|               3. Definitions.                                    |
|                                                                  |
|03000 03002 03003 03034                                           |
|__________________________________________________________________|

scr # 3002
|                                                  Index of Topics|
|        1. Power Supply                                           |
|        2. Serial Connector                                       |
|        3. Jumpers                                                |
|        4. EPROM Types                                            |
|        5. Buffers                                                |
|        6. How to Program an EPROM                                |
|        7. How to Copy an EPROM                                   |
|        8. Converting Files                                       |
|        9. Diagrams                                               |
|                                                                  |
|03000 03026 03014 03013 03004 03017 03005 03041 03008             |
|__________________________________________________________________|
```

done by adding a variable to keep track of the number of jumps and by pushing the jumped-from block number to the return stack. This increases the flexibility of the Help system and also its complexity. It may be worth it.

A compromise is to hard-code the return path. Since we may not wish to use choice zero, we dedicate it to the return screen. Whenever Esc is pressed, the zero-choice jump is made, if it exists. If it does not exist, the Help system is exited. Thus, no matter how the user gets to the current screen, the back-out path is fixed. If there are just a few screens that should return in one of several different ways, depending on how they were reached, it would be possible to duplicate these screens and give each one a different return destination. This is the method I have used. Figure Two shows the alternate definitions to do this and Figure Three shows part of the Help system text from the EPROM project.

Conclusion

So, there you have it: an easy way to add HELP to your own applications, and an illustration of the convenience of Forth blocks.

Among other things, the author makes EPROM programmers affordable by everyone. If you are interested in the EPROM Programmer kit, from plans to partial kit to fully assembled, send a S.A.S.E. to him at 809 W. San Antonio Street, San Marcos, Texas 78666 for a flier.

(Letters, from page 5.)

A problem that is exercising me now is how to write a version which does not need the number of pages of RAM to be predetermined, but constructs the necessary DOES> or ;CODE routine for each new page as it is encountered. Any suggestions?

Yours sincerely,
Philip S.H. Preston
20 Warren Street
London W1P 5DD England

Building on Monumental Analysis

Dear Sir,

I'm impressed with Guy Kelly's monumental analysis of the performance of multiple Forths (*FD* XIII/6). I suspect that his conclusion is right. There isn't enough performance difference between Forths to justify using performance as a discriminant between versions.

Guy is to be complimented on catching the timing implications of the 80x86 making byte accesses at odd addresses and word accesses at even addresses. It's documented by Intel, but not conspicuously. And it's easy to forget.

A couple of points...

1. There are obvious typos in Guy's charts. Zen Sieve performance lost a digit between Table Two and Table Three. riFORTH is shown as faster when making calls within a loop than when running the same loop empty.

Others will build on Guy's work. Future workers could probably use an errata sheet.

2. The Thread and Nest columns are more meaningful if the time in the Empty column is subtracted.

3. There is a risk that Sieve is not a typical Forth program. One wonders why the low-level speed superiority of riFORTH doesn't translate to clearly superior Sieve perfor-

(Help Engine code, continued.)

```
scr # 3003
|                                                              Overview|
|       The Bare Bones EPROM Programmer is a simple, menu              |
|driven system for programming small EPROMs.  It consists of a         |
|programmer board and software that runs on an MS-DOS type             |
|computer.  You must supply the proper voltages and, for heavy         |
|use, you will want to plug a zero-insertion-force socket into         |
|the machined-pin socket.  The programmer board connects through       |
|a serial cable to COM1 or COM2 of an MS-DOS computer.  A menu         |
|system allows you to read, erase, verify, examine, and program        |
|EPROMs.  Two jumpers to the right of the EPROM must be set for        |
|the proper EPROM type.  The menu system allows you to load            |
|from, and to save to, MS-DOS files and to convert to and from         |
|various formats (Intel Hex, and Motorola S1-S9), and even to          |
|split a file into even and odd bytes.                                 |
|                                                                      |
|03001                                                                 |
|______________________________________________________________________|

scr # 3004
|                                                        Types of EPROMs|
|                                                                       |
|The Bare Bones EPROM Programmer programs these four types of           |
|EPROMs:                                                                |
|                                                                       |
|                    2716 ( 2K x 8)                                     |
|                    2764 ( 8K x 8)                                     |
|                   27128 (16K x 8)                                     |
|                   27256 (32K x 8).                                    |
|                                                                       |
|These four types will meet the needs of the majority of small          |
|microcomputer systems.                                                 |
|                                                                       |
|                                                                       |
|                                                                       |
|03002                                                                  |
|_______________________________________________________________________|

scr # 3005
|                                              How to Program an EPROM|
|                                                                       |
|  1. Turn off power to the board (Vpp first, then Vcc).                |
|  2. Set the 2 jumpers on the board for the current EPROM type.        |
|  3. Select the EPROM type and programming parameters.                 |
|  4. Copy the Input File to the File Buffer.                           |
|  5. Insert the EPROM.                                                 |
|  6. Connect power to the board (Vcc first, then Vpp).                 |
|  7. Re-initialize the board (main menu 23).                           |
|  8. Verify EPROM is erased.                                           |
|  9. Program and verify EPROM.                                         |
|                                                                       |
|                                                                       |
|                                                                       |
|                                                                       |
|03002 03026 03013 03006 03007 03022 03026 03023 03032 03033           |
|_______________________________________________________________________|

scr # 3006
|                 Select the EPROM type and programming parameters      |
|                                                                       |
|  1. Select the EPROM type (main menu 1).                              |
|  2. Select pulse width (main menu 2).                                 |
|  3. Select maximum number of pulses (main menu 3).                    |
|  4. Set start, end, # with main menu 4-6 or 7-9.                      |
|                                                                       |
|                                                                       |
|                                                                       |
|                                                                       |
|                                                                       |
|                                                                       |
|03005 03004 09999 09999 03010                                          |
|_______________________________________________________________________|
```

```
scr # 3007
                          Copy the Input File to the File Buffer
  1. Select the Input File (main menu 10).
  2. Copy the Input File to the File Buffer (main menu 12).


03005 03023 03017
```

```
scr # 3008
                                                          Diagrams
  1. Component (top) View of Board
  2. EPROM Socket
  3. The Power Connector


03002 03009 03011 03012
```

```
scr # 3009
                                    Component (top) View of Board
  ____________________________      TTT = power connector
 |        TTT                 |     hhh = 16pin 4049 IC
 |   (Gnd,Vcc,Vpp)            |     mmm = 40pim microprocessor IC
 |  a  hhh    mmmmmm          |     eee = 28pin empty EPROM socket
 |  R  hhh    mmmmmm   (top)  |     R = red LED
 |  G  hhh    mmmmmm   eeeeee j     G = green LED
 |            mmmmmm   eeeeee j     a,b,c,d,f,g,i = resistors
 |     bcd n  mmmmmm   eeeeee       n = capacitor
 |            mmmmmm r eeeeee k     xtl = ceramic resonator
 |  f         mmmmmm r eeeeee k     SS = 9pin serial connector
 |   g        mmmmmm r eeeeee       rrr = resistor network
 |  SS        mmmmmm r eeeeee       jj = pgm/Vcc jumper
 |  SS        mmmmmm r eeeeee       kk = A11/Vpp jumper
 |  SS     i xtl       (bottom)
 |____________________________|
03008
```

```
scr # 3010
                                                  Set start, end, #

Main menu 4, 5, & 6 (for hex) or main menu 7, 8, & 9 (for
decimal)  let you specify the starting address, ending address,
and number of bytes to program.  This lets you program just part
of the EPROM if you wish.


03006
```

```
scr # 3011
The EPROM Socket is the 28-pin socket on the right edge of board
  1.--\/--.28                                        1 not    28
  2|       |27    <--- Here a 28-pin EPROM           2   used 27
  3|       |26     fills the entire socket.          3.--\/--.26
  4|       |25                                       4|       |25
  5|       |24                                       5|       |24
  6|  __   |23             Here a 24-pin EPROM ---> 6|       |23
  7| |  |  |22             fills only the lower      7|  __   |22
  8| |__|  |21             24 pins, leaving 4        8| |  |  |21
  9|       |20             empty pins at the top     9| |__|  |20
 10|       |29             of the socket            10|       |19
 11|       |18                                      11|       |18
 12|       |17                                      12|       |17
 13|       |16                                      13|       |16
 14|_______|15                                      14|_______|15
03008
```

mance. Is this a general phenomenon, or is it due to some feature of Sieve or a correctable weakness in riFORTH?

An alternative to measuring Forth performance is to calculate it on paper. A clear understanding of the operation of the Intel Bus Interface Unit look-ahead queue is needed, but I recently ran tests on a 4.77 MHz V-20 which showed hand-computed values to be within 10% of actual measured values to direct threading (4% low) and for the Laxen and Perry Virtual Machine (7% low for colon words, right on for code words). With more care and some cross checking, the results would probably be better. Hand computation of times is *much* easier than explicit testing.

Sincerely,
Don Kenney
625 Kings Way
Canton, Michigan 48188

The Future of Forth & the Forth Interest Group

Dear FIG,

Forth is, in my opinion after careful language comparisons, the best tool for intelligence computing: neural network simulators, natural language translation software, object-oriented knowledge bases with artificial intelligence functions, multi-media function coordination, etc.

Here are some suggestions, from the point of view of some Forth users, for how to increase the relative importance of Forth in the language market:

Programming language market situation

The overall distribution volume is currently declining for all programming languages, including C. In the beginning of computing, close to 100% of computers sold were used at

(Continued on page 18.)

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

Principles of Metacompilation

B.J. Rodriguez
Hamilton, Ontario, Canada

A. Introduction

This paper describes the workings of Forth metacompilers, and one example, the T-Recursive Image Compiler, in particular.

The paper will focus on:

- how to use the Image Compiler
- internal workings of the Image Compiler
- alternative approaches that have been or could be used

In this paper, "metacompiler" shall refer to Forth metacompilers in general. "Image Compiler" shall refer to this particular implementation.

1. Why Metacompile?

There are three things a metacompiler can do, that Forth's ordinary compiler can't:

a) Re-creation of the Forth kernel. The "nucleus" of Forth is inviolate once Forth is running. To improve or alter this nucleus, a metacompiler must be used.

b) Cross-compilation. Ordinary Forth produces code for the machine on which it is running. To create a Forth kernel for a different processor, a metacompiler must be used.

c) Optimization of embedded applications. The Forth compiler adds quite a lot of overhead to the kernel. It is often desirable to dispense with the unneeded code and data in memory-limited embedded applications. To compile Forth without this "excess baggage," a metacompiler must be used.

2. Objectives of the Image Compiler

The Image Compiler project had three driving goals:

a) To recompile the Forth kernel. The reason for being, and the ultimate test, of any metacompiler is the recompilation of a full Forth kernel.

b) To compile an application of any complexity. Many metacompilers can only compile a Forth kernel, perhaps with simple extensions. The Image Compiler should cope with all of the constructs which may appear in a Forth application; particularly, vocabularies and defining words.

c) To appear as close to "normal" Forth code as possible. Many metacompilers require a special syntax or lexicon, or require explicit patching instructions, in the source code. The Image Compiler should accept ordinary Forth source code as far as possible. When exceptions to this rule are required, they should resemble "normal" Forth as closely as possible.

B. A Session with the Image Compiler

To give a basic familiarity with the working of the Image Compiler, this section describes a typical session. The example used here is the metacompilation, on the IBM PC, of a new Forth kernel for the Zilog Super8 microprocessor.

First, from DOS, the Forth environment is loaded and executed. The Image Compiler (currently) uses the file-based version of real-Forth for the IBM PC:

```
A:\> RFF
real-FORTH 1.3  2  7 88
```

The basic real-Forth utilities (editor, 8086 assembler, etc.) are then loaded from screen three. This is a "master load screen" which itself will cause many other Forth screens to be loaded.

```
3 LOAD

real-FORTH on MS-DOS
(c) 1978-88 by Charles Curley
6BE7 Bytes (hex) Dictionary space available.
iok
```

This creates the normal Forth programming environment. To access the Image Compiler, real-Forth must be directed to the MS-DOS file which contains its screens:

```
USING SUPER8.SCR
iok
```

This file contains the Image Compiler, a Forth cross-assembler for the Zilog Super8, and the source code for the

Super8 Forth kernel. All of the compiler action is performed by loading the "Super8 master load screen":

```
64 LOAD
```
...very many messages, as each screen comprising the image compiler and the Super8 Forth is loaded.

At the conclusion, a copy of the complete Super8 Forth is stored in the PC's memory. Since the download utility supplied by Zilog runs independently under MS-DOS and requires an Intel format hex file, it is necessary to write this Super8 Forth image out to an MS-DOS file:

```
HEX 8000 2000 HEXFILE SUPER8.HEX
```

We must, of course, know from prior experience in setting up the source code, that the Super8 Forth's origin is 8000 hex, and that it is less than 2000 hex (8K) bytes long. (The actual origin and length can be derived from the Image Compiler, but that only serves to confuse the issue at the moment.)

real-Forth is exited by invoking the native "monitor" program (in this case, MS-DOS):

```
MON
A:\>
```

Now the Zilog development utilities can be invoked and the hex file downloaded to the target hardware, as described in the Zilog documentation.

C. The Working Environment

The computer which will do the work of metacompilation is the Host machine.

The computer whose Forth we are creating is the Target machine. The Target machine need not be physically present or connected to the Host machine.

It is possible that the Host machine is also the intended Target for the new Forth. One could use the Host's "old" version of Forth, with a metacompiler, to create a "new" Forth for the same computer.

1. The Host Machine

The Host machine must already have a working version of Forth, and any programming tools (e.g., editors) that are desired. It must have enough RAM to load the Image Compiler and the data structures it builds.

a) memory usage

In addition to space for Forth and the programming tools, the Host must have enough RAM for the Image Compiler and the data structures it builds. This amounts to about six Kbytes, plus roughly 16 bytes per metacompiled word.

The source code for the Target machine will reside on the Host's mass storage, so sufficient disk space must be available in the Host.

The output of the Image Compiler—the binary "image" of the Target code—may be directed to either RAM or disk. Their sizes must be adjusted accordingly.

b) vocabularies

The Image Compiler will reuse the names of many words in the Forth kernel. To avoid confusion, these are placed in a separate Forth vocabulary.

If the application being metacompiled is a new Forth kernel, the names will be reused yet again. These are placed in another vocabulary.

Finally, for each vocabulary defined in the Target's code, a new vocabulary will be needed in the Host.

The Image Compiler assumes that the Host's Forth supports an expandable vocabulary tree (as fig-Forth and real-Forth do).

2. The Target Machine

The Target machine will not run any code during the metacompilation process. It exists only as an abstraction. It is possible to metacompile for a processor whose hardware does not yet exist (although this poses a problem for testing).

a) memory usage

The Target will have enough RAM to run the final, metacompiled application. If the application is a new Forth kernel, the memory requirement will probably be on the order of eight Kbytes.

D. Accessing the Memory Space

The metacompilation will create a binary image of the Target code. The Host needs to be able to both write and read this "Target image."

Figure One illustrates the Host machine's memory space, and the image of the Target memory, for 16-bit Host and Target. (Host and Target both address 64K directly.)

1. Use

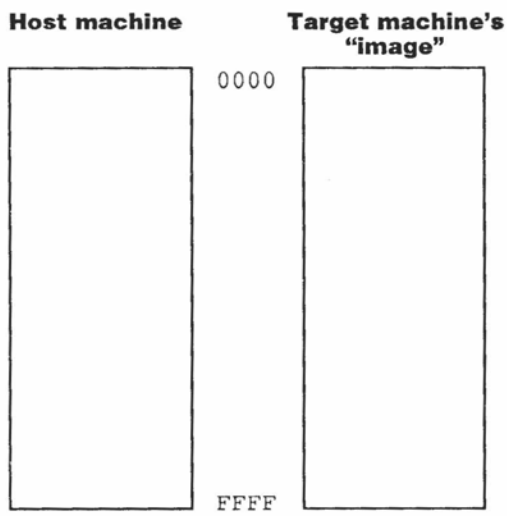
A new set of operators, analogous to Forth's memory reference operators, access the Target image:

```
T@      ( a -- n )
```
Given an address in the Target image, returns the cell (processor word) stored there. Analogous to `@`.

```
T!      ( n a -- )
```
Stores the cell value n at address a in the Target image. Analogous to `!`.

```
TC@     ( a -- c )
```
Given an address in the Target image, returns the byte stored there. Like `C@`.

Figure One. The target image.



`TC!` (c a --)
Stores the byte value c at address a in the Target image. Like `C!`.

`T,` (n)
Stores the cell value n at the "next" location in the Target image, as determined by the Target's dictionary pointer. Like `,`.

`TC,` (n)
Stores the byte value c at the "next" location in the Target image.

These last two operators imply the existence of a Dictionary Pointer for the Target image, just like the Dictionary Pointer for the Host's Forth dictionary.

`DP` (-- a)
Returns the storage address of the Target's Dictionary Pointer.

`HERE` (-- a)
Returns the contents (current value) of the Target's Dictionary Pointer.

(The Target's Dictionary Pointer words have the same names as their Host Forth counterparts. They are stored in a separate vocabulary.)

`HOST`
Selects the vocabulary containing the Image Compiler, and all of the words to access the Target image.

`NATIVE`
Selects the "native" Forth vocabulary in the Host machine.

Thus, to get the contents of the Target image Dictionary Pointer, use `HOST HERE` and to store a new value use `HOST DP !`.

It may seem strange to say `HOST` when the Target image operators are desired, or to say `NATIVE` as a synonym for `FORTH`. The reason will become clear later, when other uses for the names `TARGET` and `FORTH` will be found.

It would have been possible to re-use the names of the fetch and store operators, since they are defined in a separate vocabulary. However, "native" and "Target" fetches and stores are so frequently mixed that the constant switching of vocabularies would be cumbersome.

2. Implementation

The Target memory references are factored into a small set of Forth words. This allows different representations of Target memory to be easily implemented.

a) In Host RAM. The Target image is contained in an array in the Host's memory. On a 64K machine, this usually restricts the Target image to 8K or so.

In this (IBM PC) implementation, the Host Forth occupies a 64K segment, and the Target Image occupies a different 64K segment. `CODE` words are defined to fetch and store to the image segment.

b) On Host disk. The Target image is stored in a contiguous series of Forth disk "blocks." This increases the size of the target image, at the expense of speed.

The Forth virtual memory word `BLOCK` is used to build the image fetch and store words. Note that a cell reference must be built out of byte references, to avoid problems with cells crossing block boundaries.

c) By remote access to the actual Target system. The Target machine is equipped with a simple resident monitor program that allows memory to be examined and modified over a serial (terminal) port. This is connected to the Host machine so that the image fetch and store operators actually use the Target memory. This offers the potential for interactive debugging, at the cost of still more speed and increased volatility. [11]

3. Issues

a) Byte swapping.

When the Host and Target processors are dissimilar, it may be the case that the byte ordering they use for word storage is different. For example, the IBM PC's 8088 stores 16-bit values in memory as two bytes, low byte first; while the Zilog Super8 stores 16-bit values, high byte first. When this occurs, we refer to

the Target as being "byte-swapped" relative to the Host.

Byte-swapping can be handled transparently, by embodying this machine-dependency in the target memory reference operators. For example, one version of `T@` can be written for a byte-swapped Target, and another `T@` for a Target whose byte ordering is the same as the Host's. (The byte operators, `TC!`, `TC@`, and `TC,` are unaffected, of course.)

This does require some discipline on the part of the programmer, to use the memory reference word which is appropriate to the data being stored. This means avoiding programming "tricks" like using `T!` to store two byte values in sequence. Such tricks are rarely portable, anyway.

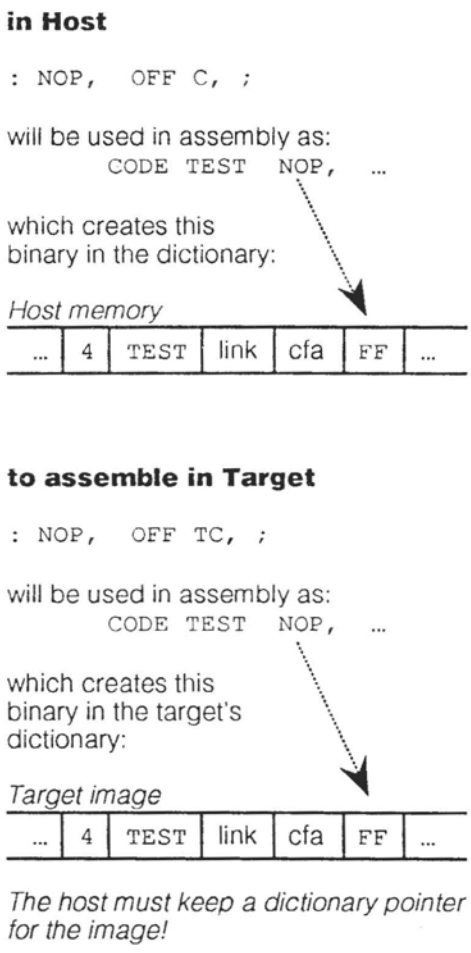
b) Different word sizes.
Another problem occurs when compiling to a machine with a larger word size, e.g., creating a 32-bit 68000 Forth on a 16-bit IBM PC Forth. The Host machine must deal with Target addresses, which of course require the larger word size. Unfortunately, there is not yet an elegant solution for this.

One approach which has been used is to employ the Host's double-precision (double-word) operators to manipulate Target data and addresses. This works, but it requires a lot of work on the part of the programmer, and it violates the objective of making the metacompiled source code look identical to "normally compiled" source code. And, as any Forth programmer can attest, mixing single- and double-precision operators on the stack is a considerable headache.

Another solution, which at least preserves the appearance of the source code, is to restrict the metacompiler to a 64K segment of the Target space. A new metacompiler variable is defined, which represents the high 16 bits of all Target addresses. Then, all of the memory reference operators add these bits to Target addresses. This adds some complexity to the compiler, since it needs to know if a value being appended to the dictionary is a Target address, which requires the addition of the high bits, or a data value, which does not.

Note that there is no problem when the Target word size is smaller than the Host's. (This, perhaps, argues for the use of a 32-bit host for metacompilation.)

Figure Two. Assembly.



E. Cross-Assembling

The most basic metacompiling function is cross-assembly. All Forth kernels and applications must begin with a series of assembly language "primitives." Fortunately, the cross-assembler is the simplest part of a metacompiler, and leads naturally to the "higher-level" functions.

1. Use

As in normal Forth, the assembler is stored in a separate vocabulary, invoked with the word `ASSEMBLER`.

Note, however, that there are now two assemblers present in the Host computer: the Host's own "native" assembler, probably loaded as part of its Forth package, and the new cross-assembler for the Target machine. The latter is distinguished by making it part of the metacompiler's "branch" of the vocabulary "tree." Thus:

`NATIVE ASSEMBLER`
Invokes the Host machine's assembler.

`HOST ASSEMBLER`
Invokes the cross assembler, i.e., the assembler which is "hosting" the metacompilation.

All assemblers are processor specific, so further details on the use of the assembler are presented elsewhere. [1,2] Suffice it to say that all of the assembler constructs and operators look exactly the same in the cross-assembler as they would in a resident assembler for the same CPU.

2. Implementation

Forth assemblers define a large vocabulary of "opcode words," whose function is to compile an opcode and its operands into memory. [1]

A simple example is shown in Figure Two. The assembler word `NOP,` compiles a `NOP` opcode, in this case for the Zilog Super8, into memory. If this were a resident assembler running on a Super8 system, the word `C,` would be used to append the assembler output to the Forth dictionary.

In order to change this to a cross-assembler, it is only necessary to change `C,` to `TC,`. Rather than append to the Host's dictionary image, the opcode word will append to the Target's. (Recall that the Target has its own Dictionary Pointer.)

Of course, an assembler for the Target processor must already have been written. Then the following substitutions can be made:

```
C,     becomes   TC,
,      becomes   T,
C@     becomes   TC@  for accesses to assembled code
@      becomes   T@     "     "     "     "     "
C!     becomes   TC!    "     "     "     "     "
!      becomes   T!     "     "     "     "     "
HERE   becomes   HOST HERE
```

Note that only certain fetches and stores—those which access the machine code being assembled—are substituted with the Target equivalents. Since the assembler itself is still resident in the Host machine, any fetches and stores to its data tables or to get parameters stored with defined words, must use the "native" fetch and store words.

It is becoming quite common for all Forth assemblers to be written using the "T-prefix" words to access the assembled image. This makes the assembler "metacompiler ready." For resident assembly, it is straightforward to define the words `TC,`, `T,`, etc., as synonyms of their "normal" counterparts.

3. Issues

a) Assembly structures

Assemblers written in Forth are usually "structured assemblers," providing assembly level `IF...THEN...ELSE`, `BEGIN...UNTIL`, and similar constructs. These are usually implemented by stacking addresses—either branch destinations, or branch arguments to be patched—while the structure is being parsed, and resolving all of these addresses when the structure is concluded.

Naturally, in a cross-assembly environment, the addresses being stacked and resolved are Target addresses, and the branches being patched are in the Target memory space. The author of a cross-assembler must keep this in mind, and use the appropriate Target memory operators when defining the structure words for the assembler.

b) Labels

Forth assemblers typically discourage the use of labels, for two reasons. First, every definition in a Forth environment takes up space in the compiled code; assembler labels are generally meaningless after the assembly is complete, and would simply waste dictionary space. Second, all Forth definitions are appended to a single dictionary, which raises the interesting problem of how to append a definition for an assembler label, while still only halfway through a definition for a `CODE` word.

Several solutions have been offered for "normal" Forth environments, including predefined "local" labels with fixed names, and a second "disposable" dictionary space.

In the metacompiled environment, however, these two problems do not arise if the label is defined in the Host's dictionary. Such a label consumes no space in the Target, and does not disrupt the half-formed `CODE` definition in the Target image. Of course, the label is only available during the metacompilation.

Traditionally, labels and equates are defined with the phrase

`EQU name  ( n -- )`
Defines an assembler "equate" having the value n. When `name` is used during assembly, n is placed on the stack

To mark a point in the assembler code with a symbolic label, use
`HERE EQU name.`

This is exactly equivalent to the line:

```
name: .equ $
```
in a "conventional" assembler.

Note the distinction between `EQU` and `CONSTANT` in the Target code. An `EQU` occupies no space in the Target, can be used within an assembler definition, and is only available during metacompilation. A `CONSTANT` is a definition in the Target dictionary, cannot be created within an assembler definition, and will still be present when the Target system is executed

Text and figures continue in the next issue.
Code begins in next issue.

(Letters, from page 11.)
least occasionally for programming. Now fewer than 1% are used for programming.

This fundamental change of user behavior affects all programming languages. The frequent version changes in the C market (e.g., additional prices for minor improvements) are typical of the final phase of a saturated market.

When evaluating all available information, I suppose that Forth lost ground between 1985 and 1989, relative to other languages. But I have the impression that since 1989, Forth has advanced compared to LISP, Prolog, Pascal, and BASIC. My impression is mainly based on the boost given to Forth by powerful shareware versions.

My evaluation includes the European markets, where Forth now advances compared to other languages. Only C is doing better—and not because of quality but because of having been chosen by IBM and for Unix.

Forth power creates market problems

Forth is not only a programming language, but a programming language generator and a compiler generator. While the number of people willing to study a programming language steadily decreases, the number of people willing to study the vaster complexity of a compiler generator decreases even more.

But interest in such computing power is essential to the future of Forth. It is what makes the difference when comparing Forth to C and Pascal. So a very widespread use of Forth should not be considered to be the most important goal. Improving Forth's computing power is more important, so that experienced Forth users will continue to prefer Forth, will have market access thanks to Forth, and in this way will be willing to finance the future evolution of Forth.

The increasing complexity of hardware and operating systems requires Forth to increase in complexity, which may further reduce its appeal to the occasional user and which may increase its appeal to the professional user.

fig-Forth and Forth Dimensions

In a time of shrinking software markets, many Forth programmers live on shrinking budgets. It is important now to remember from whom we received the most information and support in the past. Forth is, for many programmers, the most important tool for open, unlimited, and honest access to the computer.

This is mainly due to fig-Forth and *Forth Dimensions*, and to those doing the work there for so many years. So Forth programmers should, in these times, try to increase their support for the Forth Interest Group to help compensate for the Forth programmers who have left the software market. Especially in this time of reduced budgets, greater tasks have to be mastered: true 32-bit computing for the PC standard; Windows; OS/2 version ≥ 2.0; and the future Power PC.

It is important, within the limits of their budgets, that Forth users:

- continue to subscribe to *Forth Dimensions*
- use the FIG disk library intensively
- pay the shareware fees for programs used
- buy from Forth vendors as much as possible

These things are necessary if Forth users want to receive Forth source code for future hardware and operating system standards.

32-bit Forth

It was always my opinion that it is a decisive error to maintain a 16-bit data stack as part of the Forth standard, with possibly identical operators for 16- and 32-bit data. This might be tolerable in a programming language limited to small application programs, but it creates important practical problems for complex systems concepts. When using all the power of Forth, this concept simply does not work. In my opinion, the only acceptable solution is a 32-bit data stack standard with different operators for 16, 64, 128, and perhaps 256 bits (256-bit operators are the key to symbolic processing, string sorting, etc. at "brain speed" on a PC).

16-bit CPUs will tend to disappear during 1992–1993. A full 32-bit Forth (dictionary and data stack) for 2 gigabytes of RAM may now become the *de facto* standard. Let's hope that the developers of F83 and F-PC can be encouraged to write new, 32-bit versions.

(For conversion to a 32-bit data stack, I can furnish the several lines to modify the interpreter and debugger. The modification even allows switching arbitrarily during compilation between 16- and 32-bit data stacks, which is useful for importing unmodified assembler subroutines originally written for a 16-bit data stack.)

Portability: PC, Power PC, Macintosh, Amiga, etc.

fig-Forth started with a high degree of portability (see the impressive list on the FIG Mail Order Form). Programmers would like to return to this with a low-priced commercial or shareware "standard" Forth, including metacompiler and 32 bits, with a completely identical wordset used by versions for the PC, Macintosh, Amiga, the future Power PC, and other major hardware.

Such portability is, for many tasks, more important than the volume of features in a Forth implementation. Some Forths complying with Forth-83 came close to this approach. Would it be useful to find all these versions (the ones that are public domain or shareware) and to include them in the FIG disk library?

For example, there are F83 implementations for some 680x0 computers. Such implementations should also be available on PC disk format, to aid those who want to observe portability issues while writing a program solely for the PC.

The presence of such new library disks could enhance a growing interest in portability and influence authors of shareware and commercial Forths.

C interface

Three types of C interfaces are on my wish list:

- A C implementation on top of Forth
- A Forth implementation on top of C

- C subroutines linkable to Forth

The problem is not so much whether this could happen, but whether it could be included in low-priced versions of Forth and up to what level of complexity.

Forth is the better programming philosophy; C has more subroutine libraries. Both together would be the optimum.

Interpreter/parentheses

To make Forth less strange, a formal principle should, perhaps, be sacrificed: The conventional use of parentheses for arithmetical expressions should be introduced into all Forth implementations, perhaps as an optional feature (switch on/off). This would eliminate the major superficial reason why most programmers and future programmers refuse to try Forth.

Commercial Forths

Commercial versions of Forth should be available, *in addition to* powerful shareware and public-domain versions. The problem is that experienced Forth programmers hesitate to buy commercial Forths because the complete source code is not available or carries a relatively high price tag. This means they have to pay for the whole thing before they can be sure that the whole thing is, in fact, the "whole" thing.

The language market no longer promises windfall profits. The good side of this is that software source and idea piracy is no longer a major risk. Perhaps now a new vendor strategy is possible: to sell commercial Forths with complete source code, all for the low price of, perhaps, $100. Vendor revenues could be derived from specialized programs, guaranteed instantly compatible with the main system, and with a library update service for new versions of the main system.

The availability of tools for corporate users might increase this effect, e.g., network support or data acquisition as relatively expensive options. A further possibility might be to apply a low tariff on individual programmers and for prototyping, but the full commercial tariff on corporate users requiring hotline support and professional manuals.

Shareware tariff

The success of Forth shareware authors might be improved by observing some user wishes:

Shareware distribution of Forth programs should avoid complicated registration fee rules. For example, if an accompanying floating-point library is not covered by the main shareware payment, the additional fee should be clearly stated for the various possible intended uses. Forth programmers are relatively honest payers of shareware fees, but if the payment conditions are not clear, they may simply avoid integrating that routine into an application. And it may be difficult to calculate the price of a shareware subroutine because the programmer may not know all possible future uses of his new program.

Shareware versions should also avoid including major features which are not shareware and which therefore require payment for any commercially redistributed copy. Programmers will hesitate to begin using such a Forth system

because of potential copyright problems.

FIG disk library

The disk library maintained by the Forth Interest Group probably is the most important element for the future of Forth. Remember, at least 60% of Forth users are outside the United States and have no easy access to relatively complete Forth libraries on-line in the U.S. Perhaps Forth documents that were formerly printed could also be distributed on disk from now on, rather than on paper. This will be efficient soon, with the standardization of 3.5" magneto-optical disks holding 128 Mb. For example, back issues of *Forth Dimensions* and conference proceedings could be input in graphics mode (with a full-page scanner), with program source also included as ASCII files (thanks to OCR and to authors' original source disks).

With changing disk capacities, pricing based on the number of 360 Kb disks required will soon become obsolete. In the future, information will either be free or not more than the cost of the media and duplication expenses.

Forth users must leave these decisions to those doing the work: the Forth Interest Group. My opinion: it would be in the best interest of Forth users if FIG were to significantly increase the volume of its library offerings, forming the most important Forth library anywhere. This may require that most library items not be public domain, but distributed exclusively by the Forth Interest Group, in order to pay for the important work of, for example, scanning former printed documents into disk files.

Conclusion

I hope my miscellaneous proposals and observations, from the point of view of an active Forth user, include helpful items for those whose decisions will affect the future of the language. P. Kahn (of Borland) chose not to distribute Forth "because it is not a programming language but a religion."

Why not? Forth programmers know a higher truth: businesses live and die, only religions survive for thousands of years.

Peter Roeser
SOFTEXT
Paris, France

Pattern Matching Rings True

Ariel Scolnicov:

Your article, "Structured Pattern Matching" (*Forth Dimensions* XIV/2) arrived just as I was looking for such a device—thank you! I had just finished reading a chapter that you will want to research yourself in *Software Tools* (Kernighan and Plauger; Addison-Wesley, 1976). It presents one of the few public discussions of regular expressions and pattern matching. The chapter "Text Patterns" shows how to parse and match regular expressions.

Their "amatch" routine is what you are looking for. But they leave unsolved your alternation and concatenation operators. I salute you and eagerly await the code. *[Printed in this issue. —Ed.]* With your word synopsis and design description, I may be able to forge an implementation. There's no question in my mind that you have the right design, in terms of:

- state machine
- "parse" tree
- action and private data

I'm going to take the time to study your design and try to understand its subtleties (`MANY`, `MOST`, ...). As a member of the technical staff at Bell Labs, I'm among the minority who advocate Forth. Like yourself, I too can use encouragement as I am working to make a copy of ANS Forth available within the Labs.

Your words will be valuable in that effort, as they go a long way to implement Unix System Tools functionality. They allow building replacements for *grep* and *sed* (and *awk*!).

Much of my tool work is in shell, using these utilities; I'm trying to convert entirely to Forth. Lack of string-matching tools of the sort you present has hindered my work.

Congratulations, and thank you.

Respectfully,
Marty McGowan
24 Herning Avenue
Cranford, New Jersey 07016-1946

Advertisers Index

Colour Vision Systems ... 21
The Computer Journal .. 20
FORML Conference 44
Forth Interest Group .....
.....centerfold
Harvard Softworks 12
Miller Microcomputer
Services 21
Saelig Company 19
Silicon Composers 2

Structured Pattern Matching

Ariel Scolnicov
Mevasseret Zion, Israel

This code implements the ideas discussed in the author's article, "Structured Pattern Matching," which appeared in our last issue. Ask the Forth Interest Group office about the availability of back issues (see order form, this issue). This code is also available for downloading from the Forth software library maintained on GEnie's Forth RoundTable.

Pattern Matcher Logic Engine AS 10/1/92

```
{
    anew logic-engine
}

The logic engine consists of all universal pattern matching facilities, i.e.
it doesn't have anything to do with the string itself.

Two stacks are used: the call stack is used to store return addresses for
pattern calls; the backtracking stack is used to store the current state at
every backtrack point. We define the two stacks to use the same area of
memory, one growing upwards and the other downwards. If complex patterns are
matched, STACK-SIZE may need to be increased.
{
    1024 constant stack-size
    create stack-start stack-size allot
    stack-start stack-size + 2- constant stack-end
    variable call-sp                \ Stack pointer
    variable back-sp                \ Stack pointer

    : init-call     \ Initialise call stack
      stack-start call-sp ! ;
    : init-back     \ Initialise backtrack stack
      stack-end back-sp ! ;

    : ?call         ( -- flag ; True iff stack empty )
      call-sp @ stack-start u<= ;
    : ?back         ( -- flag ; True iff stack empty )
      back-sp @ stack-end u>= ;

    : ?callempty    \ Check if call stack empty )
      ?call abort" CALL stack empty!" ;
    : ?backempty    \ Check if backtrack stack empty )
      ?back abort" BACKTRACK stack empty!" ;

    : ?stacksfull   \ Check if both stacks full )
      call-sp @ back-sp @ u> abort" CALL/BACKTRACK stacks full!" ;

    : >call         ( n -- ; Push to call stack )
      ?stacksfull
      call-sp @ !
      2 call-sp +! ;
    : call>         ( -- n ; Pop call stack )
      ?callempty
      -2 call-sp +!
```

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
Last 50 Introduction to FIG, threaded code, TO variables. fig-Forth.

Volume 2 Forth Dimensions (1980–81) — 102 - $15/16/18 — OUT OF PRINT
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
Last 50 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 — OUT OF PRINT
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 — OUT OF PRINT
Computer graphics, 3D animation, double-precision math words, overlays, recursive sort, a simple multi-tasker, metacompilation, voice output, number editor, menu-driven software, vocabulary tutorial, vectored execution, data acquisition, fixed-point logarithms, Quicksort, fixed-point square root.

Volume 6 Forth Dimensions (1984–85) — 106 - $15/16/18
Last 100 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
Last 100 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
Last 100 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
Last 100 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
Last 100 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
Last 100 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
Last 100 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
Last 50 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
Last 50 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
Last 100 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
Last 100 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
Last 100 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 — OUT OF PRINT
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
Last 100 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.

Fax your orders 510-535-1295

1987 **FORML PROCEEDINGS** 317 - $40/43/50
Last 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 - $40/43/50
Last 100 Includes 1988 Australian FORML, Human interfaces, simple robotics kernel, MODUL Forth, parallel processing, programmable controllers, Prolog, simulations, language topics, hardware, Wil's workings & Ting's philosophy, Forth hardware applications, ANS Forth session, future of Forth in AI applications.

1989 **FORML PROCEEDINGS** 319 - $40/43/50
Last 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
Last 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 ..."

SCIENTIFIC FORTH, Julian V. Noble 250 - $50/52/60
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), **IBM**

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 (hard-cover 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.

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

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 V4.4, Mark Smiley C203a - (3)
NEW 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 V2.2NEW, 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** NEW

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**

Going out of Print !!!

Back issues of *Forth Dimensions*
FORML Conference Preceedings

If you haven't got a complete set of *FORML*s and *Forth Dimensions,* you had better act now. They will not be reprinted in the future.

See the *Forth Dimensions Article Reference* and the *FORML Article Reference* for indexed details of articles that you are looking for.

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

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

6502	514 - September 80	**9900**	519 - March 81
6809	516 - June 80	**Apple II**	521 - August 81
8080	517 - September 79		

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 _____
email _____ Fax _____

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 Expiration Date _____
Card Number _____
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 by surface, other methods available by special request. 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.

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

```
        call-sp @ @ ;
    : !call         ( n -- ; Change top of call stack )
        ?callempty
        call-sp @ 2- ! ;

    : >back         ( n -- ; Push to backtrack stack )
        ?stacksfull
        back-sp @ !
        -2 back-sp +! ;
    : back>         ( -- n ; Pop backtrack stack )
        ?backempty
        2 back-sp +!
        back-sp @ @ ;
}

The auxiliary stack is never used by the logic engine or the matching unit.
It is reserved for use by the pattern. It is saved when backtracking.

{
    64 constant aux-size            \ Small stack
    create aux-stack aux-size allot
    variable aux-sp                 \ Stack pointer

    : init-aux      \ Initialise stack
        aux-stack aux-sp ! ;
    : init-stacks
        init-call   init-back   init-aux ;

    : ?aux          ( -- flag ; True iff stack empty )
        aux-sp @ aux-stack u<= ;
    : ?auxempty     \ Check if auxiliary stack empty
        ?aux abort" Auxiliary stack empty" ;
    : ?auxfull      \ Check if auxiliary stack full
        aux-sp @ aux-stack aux-size + u>=
        abort" Auxiliary stack full" ;

    : >aux          ( n -- ; Push to auxiliary stack )
        ?auxfull
        aux-sp @ !
        2 aux-sp +!
        ;
    : aux>          ( -- n ; Pop from auxiliary stack )
        ?auxempty
        -2 aux-sp +!
        aux-sp @ @
        ;
}

The state pushed onto the backtrack stack consists of the state variables'
area the call stack and the auxiliary stack. The last two are of variable
size, so we also push the sizes of the two stacks onto the backtrack stack.

{
    variable state-adr
    variable state-len              \ Location of state area

    : back>cmove    ( adr len -- ; Pop block from backtrack stack )
        tuck back-sp @ 2+ -rot              ( len back adr len )
        cmove                               ( len )
        back-sp +!                          ( )
        back-sp @ stack-end u>              \ Can't use ?backempty here
        abort" Backtrack stack underflow!" ;

    : back>state    ( -- ; Pop state off backtrack stack )
        \ Assumes action address already popped
        aux-stack back>                     ( aux-stack aux-sz )
        2dup back>cmove                     \ Pop auxiliary stack
        ( aux-stack aux-sz ) + aux-sp !     \ Restore aux-sp
        stack-start back>                   ( stack-stack call-sz )
        2dup back>cmove                     \ Pop call stack
```

```
        ( stack-start call-sz ) + call-sp ! \ Restore call-sp
        state-adr @ state-len @ back>cmove  \ Pop state area
    ;

    : back>drops    ( -- ; Drop state off backtrack stack )
        \ Assumes action address already popped
        back> dup                           ( aux-sz aux-sz )
        aux-stack + aux-sp ! back-sp +!     \ Set aux-sp and drop items
        back> dup                           ( call-sz call-sz )
        stack-start + call-sp ! back-sp +!  \ Set call-sp and drop items
        state-len @ back-sp +!              \ Drop state area
        back-sp @ stack-end u> abort" Backtrack stack underflow!"
    ;

    : cmove>back    ( adr len -- ; Push block to backtrack stack )
        dup negate back-sp +! ?stacksfull   ( adr len )
        back-sp @ 2+ swap                   ( adr back len )
        cmove ;

    : state>back    ( -- ; Push state to backtrack stack )
        state-adr @ state-len @ cmove>back  \ Push state area
        stack-start call-sp @ over - tuck   ( call-sz adr call-sz )
        cmove>back                          \ Push call stack
        >back                               \ Push call stack size
        aux-stack aux-sp @ over - tuck      ( aux-sz adr aux-sz )
        cmove>back                          \ Push auxiliary stack
        >back                               \ Push aux stack size
        ['] back>state >back                \ Action address to pop state
    ;
}
```

The driver loop for the logic engine is very simple: at every step, pop the next node's address from the call stack. If the call stack is empty, the match succeeded. The first cell in a node should be the address of an execution word for that node. Following the execution address there may be private data. The stack action of the execution address is (ADR+2 -- FLAG) where ADR+2 is the start of private data and FLAG indicates success (TRUE) or failure (FALSE). If failure is returned, a state is popped off the backtrack stack and execution continues at that state. If the backtrack stack is empty, the match failed.

```
{
    : driver    ( start-node -- flag )
        init-stacks >call                   \ Only root node on stack
        begin
            call> dup 2+ swap perform       \ Fetch and execute node
            0= if                           \ Backtrack?
                ?back if false exit then    \   none possible ==> FAIL
                back> execute               \ pop state
            then
        ?call until true                    \ Until call stack empty
    ;
}
```

We can now define the logical connectors (&&, ||), as well as some more specialised operators (FAIL, NULL, CUT, MANY, MOST). Since && and || perform far better in right-recursive situations, [& ... &] and [| ... |] build lists as [correct] right-recursive patterns. If you want patterns built as you specify them, use && and ||.

```
{
    : <&>           ( private -- flag ; Execution word for AND nodes )
        2@ swap >call >call true ;          \ Push both subtrees
    ' <&> constant '<&>

    : <|>           ( private -- flag ; Execution word for OR nodes )
        2@ swap >call state>back            \ Save backtrack point
        !call true ;                        \ Leave 1st subtree on call stack
    ' <|> constant '<|>
```

```
    : binary        ( exec-adr | op-name -- ; Define binary operator )
        create ,
        does>           ( a1 a2 cfa -- adr )
            here >r                         \ Save node address
            @ , swap , ,                    \ Lay down exec-adr, a1, a2 )
            r> ;                            ( adr )
    '<&> binary &&
    '<|> binary ||

    : [&    0 ;     ' [& alias [|           \ Leave sentinel value
    : linker        ( exec-adr -- ; Create a linker word )
        create ,
        does>           ( 0 a1 a2 ... an cfa -- ; Link all nodes )
            @ >r begin                      ( 0 a1 a2 ... an )
                over while                  \ While more nodes to link
                r@ execute                  \ Execute linker procedure
            repeat                          ( 0 a1 a2 ... an-1 )
            nip r>drop                      \ Drop 0 and cfa
    ;
    ' && linker &]
    ' || linker |]
}

FAIL, NULL and CUT are patterns which we call when we need. There are no
defining words for these patterns - we build them once only. Note the use of
CREATE to name a single-node pattern.

{
    : 1node         ( exec | name -- ; create 1 node pattern )
        create , ;

    : <fail>        ( private -- flag )
        drop false ;                        \ Always fail.
    ' <fail>    1node fail

    : <null>        ( private -- flag )
        drop true ;                         \ Always succeed
    ' <null>    1node null

    : <cut>         ( private -- flag )
        drop init-back true ;               \ Empty backtrack stack
    ' <cut>     1node cut
}

OPT creates an optional pattern, i.e.
    x OPT generates "x NULL ||".
This means that, if possible, the pattern WILL be matched. If you want
matching to occur only as a last resort, use "NULL x ||".

{
    : opt           ( pat1 -- pat2 )
        NULL || ;                           \ Either match or don't
}

~~ implements search negation -- it matches the least necessary to
"disprove" a given pattern. Note that this is generally quite different from
logical negation.

Implementation:
    x ~~ generates a node which, when executed, pushes NEG-ACCEPT onto the
    backtrack stack (if x failed, i.e. negation succeeded) and NEG-REJECT
    onto the call stack. NEG-ACCEPT pops words off the call stack until
    NEG-REJECT (inclusive); NEG-REJECT pops states off the backtrack stack
    until NEG-ACCEPT (inclusive), then fails.

{
    : unary     ( exec | name -- ; Create unary pattern definer )
        create ,
        does>                               ( n cfa -- adr )
            here >r @ , , r> ;
```

```
    variable 'neg-accept        variable 'neg-reject
    : <~~>          ( private -- true )
        'neg-reject   >call                 \ NEG-REJECT on call stack
        @ >call                             \ Next to execute is x
        'neg-accept @ >back                 \ NEG-ACCEPT to backtrack stack
        true ;

    : neg-accept    ( -- ; Drop from CALL until neg-reject inclusive )
        begin
            call> 'neg-reject =
        until
    ;
    : neg-reject    ( private -- false ; Drop from BACK until neg-accept )
        drop begin
            back> 'neg-accept @ <> while    \ While state stacked
            back>drops                      \ Drop it
        repeat false
    ;

    ' neg-accept 'neg-accept !          ' neg-reject 'neg-reject !

    ' <~~> unary ~~
}
```

MANY creates patterns to match several repetitions of a pattern (possibly none). It performs backtracking, so m" xy" many m" xyz" && performs as expected. Note that using MANY on long strings can use up lots of backtrack stack space.

MOST is similar, but matches as many repetitions as possible, unlike MANY, which matches the least number possible.

Implementation is very simple:
    x MANY generates "NULL x y && ||",
where y is the address of the node created by "||". Getting at y is the tricky bit. Here we use the (*implementation specific*) fact that the length of an "&&" node is 6 bytes.

```
{
    : many      ( pat1 -- pat2 ; Matches fewest )
        here 6 + && null swap || ;
    : most      ( pat1 -- pat2 ; Matches most )
        here 6 + && null      || ;
}
```

For recursive patterns we need some way of forward referencing. This is provided by @CALL, which inserts the contents of the specified variable into the call stack. The result is that you can use v @CALL in an early pattern, and later on set v to a given pattern, getting a forward reference. @CALL can probably also be used to call a different pattern every time, but this is probably being too clever.

```
{
    : <@call>   ( private -- flag ; Push contents of variable to call stack )
        @ @ >call true ;

    ' <@call> unary @call
}
```

It is very useful to be able to execute a word while matching a pattern. The word EXEC calls a given word, which should have no stack action. Use the auxiliary stack if you need temporary storage. Note that the word is executed as encountered, and backtracking may cause it to be executed several times. However, the auxiliary stack is restored through backtracking, so careful use of it should make complex actions possible. Alternatively, just store enough data to enable retrieval of information later on.

DO NOT use EXEC to call DRIVER recursively; it is not re-entrant!.

```
{
    : <exec>    ( private -- flag ; Perform stored routine )
      perform true ;
    ' <exec> unary exec
}
```

SNOBOL-style string matching using the Logic Engine

AS 10/1/92

```
{
    anew string-pattern-matcher
}

The state of the string matcher is defined by the current position in the
string. It is of course possible to store this as an offset only, but it is
faster to store an address into the string and a remaining length. Of
course, we also need to store the address and length of the entire string.
{
    variable str-adr        variable str-len    \ adr & len of string
    variable str-start                          \ True start of string
                                                \ (for PSEARCH).
    create cursor 4 allot                       \ State area
    cursor      constant cur-adr
    cursor 2+   constant cur-len
    cursor state-adr !      4 state-len !       \ Identify state area
}
The cursor always points to the current character in the string. Several
common manipulations are defined here.
{
    : init-cur      \ Initialise current position to whole string
      str-adr @ cur-adr !
      str-len @ cur-len !
    ;
    : cur-char      ( -- c ; Return current character )
      cur-adr @ c@ ;
    : advance       ( n -- ; Advance cursor by n chars )
      dup     cur-adr +!
      negate  cur-len +! ;
    : 1advance      ( -- ; Advance cursor by 1 char )
      cur-adr incr        cur-len decr ;
}
Pattern matches are handled by PMATCH ( adr len pat -- ?len flag ).
Pattern searches are handled by PSEARCH ( adr len pat -- ?adr ?len flag ).
The difference is of course that PSEARCH advances along the subject string.
{
    : pmatch        ( adr len pat -- false | len true )
      -rot str-len !                        ( pat adr )
      dup str-adr ! str-start !             ( pat )
      init-cur driver                       ( flag )
      if      cur-adr @ str-adr @ - true    ( len true )
      else    false then                    ( false )
    ;
    : psearch       ( adr len pat -- false | adr len true )
      -rot str-len !                        ( pat adr )
      dup str-adr ! str-start !             ( pat )
      begin
          init-cur dup driver               ( pat flag )
          if
              nip str-adr @ cur-adr @ over -  ( adr len )
              true exit then                ( adr len true )
      str-len @ while                       \ Until end of string
          str-len decr str-adr incr         \ Advance start
      repeat                                ( pat )
      drop false                            ( false )
    ;
}
It is of course possible to have just one primitive pattern: match 1
character. However, it is far more efficient to define M", which matches a
substring, and ANYOF", which matches if the current character is in the given
string. M' and ANYOF' are also provided, to allow the doublequotes character
```

```
to be matched.
{
    : $=            ( a1 a2 n -- flag ; Strings at a1 and a2 = for n chars? )
        dup 0< if 3drop 0 false exit then       \ Exit if n<0
        dup>r 0 ?do                             ( a1 a2 )
            over c@ over c@ <> if               \ Not equal?
                2drop i 1+ false
                undo r>drop exit                ( false )
            then
            1+ swap 1+                          \ Advance both strings
        loop                                    ( a2 a1 )
        2drop r> true                           \ Equal for n characters
    ;

    : <m>           ( private -- flag; Check substring at current position )
        count dup cur-len @ > if                ( adr len )
            \ Not enough characters left, match what there is and fail
            drop cur-adr @ cur-len @            ( adr cur-adr cur-len )
            $= drop advance false               ( false )
        else
            cur-adr @ swap                      ( len adr cur-adr len )
            $=                                  ( len flg )
            swap advance                        \ Match len characters
        then
    ;

    : <anyof>       ( private -- f; check if current char is in string )
        cur-char swap                           ( chr private )
        count 0 ?do                             ( chr adr )
            2dup c@ = if                        \ Character in string?
                1advance 2drop                  \ Match it
                true undo exit                  \ and return true
            then 1+                             \ Try next possibility
        loop
        1advance 2drop false                    \ No match
    ;

    : strpat        ( adr len exec -- pat )
        here >r                                 \ Save address
        , tuck here place 1+ allot              \ Store exec and string
        r>                                      ( pat )
    ;

    : m"        ascii " parse ['] <m> strpat ;
    : anyof"    ascii " parse ['] <anyof> strpat ;
    : m'        ascii ' parse ['] <m> strpat ;
    : anyof'    ascii ' parse ['] <anyof> strpat ;
}
POS and RPOS are used to check the cursor position. n POS succeeds only if
the cursor is in offset n; n RPOS counts characters from the end. v @POS and
v @RPOS are similar, except that n is fetched from the variable v. HEAD and
TAIL are synonyms for 0 POS and 0 RPOS, respectively.
{
    : <pos>         ( private -- flag ; Cursor at stored position? )
      @ str-start @ + cur-adr @ = ;
    : <rpos>        ( private -- flag ; Cursor at stored position from end? )
      @ cur-len @ = ;
    : <@pos>    @ <pos> ;                       \ Fetch from variable and
    : <@rpos>   @ <rpos> ;                      \ perform POS or RPOS

    ' <pos>  unary pos              ' <rpos>  unary rpos
    ' <@pos> unary @pos             ' <@rpos> unary @rpos
    0 pos constant head             0 rpos constant tail
}
EXEC isn't very useful on its own, but if we define PUSH to push the current
cursor position onto the auxiliary stack we can perform an action on a
matched substring. In addition, we define SUBSTRING to perform a specified
routine on a matched substring. The stack action of the routine is
( adr len -- ), where adr and len specify the substring. SUBSTRING
builds [& PUSH pat1 substring-node &], where pat1 is the pattern specifying
```

```
the substring, and substring-node is a <substring> node with the execution
address stored.
(
    : <push>        ( private -- flag ; Push current position to aux stack )
        drop cur-adr @ >aux true ;
    create push     ' <push> ,

    : <substring>   ( private -- flag ; Execute stored routine on )
        aux> cur-adr @ over -               ( exec-adr adr len )
        rot perform true                    \ Perform routine and succeed
    ;

    : substring     ( pat1 exec-adr -- pat2 )
        here ['] <substring> , swap ,       ( pat1 <substring>-node )
        && push swap &&                     ( pat2 )
    ;
)
```

Evaluate expressions using the Pattern Matcher

AS 1/1/92

```
anew evaluator decimal
\ Auxiliary stack routines.
: num>aux                           ( adr len -- ; Push number to aux stack )
    drop 1- 0. rot convert 2drop >aux ;
: -aux  aux> negate >aux ;
: *factor   aux> aux>       * >aux ;
: /factor   aux> aux> swap  / >aux ;
: +term     aux> aux>       + >aux ;
: -term     aux> aux> swap  - >aux ;

\ Forward reference EXPR and TERM
variable expr

    anyof" 0123456789"
constant DIGIT

( A number is a sequence of digits )
    digit digit most && ' num>aux substring
constant NUM

( A factor is a number or a bracketed expression, possibly preceded by a )
( unary negation operator.                                               )
    num     [& m" (" expr @call   m" )" &]  ||
constant    (FACTOR)

    (factor) [& m" -" (factor) ' -aux exec &] ||
constant    FACTOR

( A term is a term multiplied or divided by a factor, or a factor )
    [& factor
        [|  [& m" *" factor ' *factor exec &]
            [& m" /" factor ' /factor exec &] |] many
    &]
constant TERM

( An expression is an expression plus or minus a term, or a term )
    [& term
        [|  [& m" +" term ' +term exec &]
            [& m" -" term ' -term exec &] |] many
    &]
EXPR !
\ We can use expr @ now, because it already has a value
expr @ tail && constant exp

: eval   32 word count exp pmatch if drop aux> . else ." Failed" then ;

\s
.( 2+3*5 = ) eval 2+3*5 cr
.( 2+3*-4 = ) eval 2+3*-4 cr
```

A Forum for Exploring Forth Issues and Promoting Forth

Fast FORTHward

The Changing Marketplace

I was dumbfounded by one of the comments made by Charles Moore as reported by Jack Woehr (see the accompanying "Press Watch" story). Through their collaboration, Chuck says that Apple and IBM will both forfeit the future. Chuck is apparently critical of the complex operating systems for which these companies have gained a reputation.

What is the trend going to be? Will Forth be positioned to take advantage of it?

Elizabeth Rather offered a sobering viewpoint when she said, "...if there is a groundswell of people that are appalled by bigger-is-better, she hasn't seen it."

I am pleased to think that as Forth programmers, we abhor complexity. But despite years of resistance, I have seen myself migrate to 800K word processors, 1Mb page-layout and graphics programs, and the like. These are choices that I made reluctantly, and only as I came to understand my needs better. (I still use non-WYSIWYG and non-graphical tools on a daily basis where I work because I also understand how some of my needs cannot be addressed by the available GUI tools.)

Before I came to Apple, I felt my non-graphical tools were entirely adequate. At Apple, I started alternating between UNIX-style programs for word processing and GUI word processors. I found that when I use the non-graphical tools all day, I would go home with head hurting. That observation, made repeatedly over a period of about two years, fully convinced me that I actually do prefer a GUI tool whenever one is suitable for my work. As a user, I care less about buying more RAM memory than I do about a lifetime of headaches.

...we cannot view ourselves as iconoclasts in perpetual defiance of the large companies driving such standardization efforts.

(In a day's work session, it's not unusual for me to enter hundreds of commands. The need to run so many commands often is due to the iterative effort required to correctly specify each element in a complex UNIX command. For example, I often err because some commands count fields starting with one, and others starting with zero.)

Perhaps the ability of the GUI tools to dispense with my headaches can be explained this way: The operation of the user interface occupies the visual regions of my brain—leaving the analytical side of my brain free to think about what I am trying to accomplish, without encumbrance.

Even though I am able to use the non-graphical tools to regularly do amazing things, the process of using them is not itself a joy. The reward comes only when the result is achieved, and by that time I am often uncomfortable, tired, and ready to quit for the day. Why is that?

As human beings we appreciate rich visual stimulation. Accordingly, the GUI is here to stay. The next concern should be how Forth can be incorporated into the world of user interface objects, or how it can support such a world.

The interest in programming languages is also about to be eclipsed by the architectural initiatives being taken by companies like Apple and Microsoft. They are hard at work creating application programming interfaces (APIs) to help leverage the ability of applications and people to collaborate more effectively.

Application programming interfaces are being developed now to route documents and messages in an endpoint-independent fashion, so destinations such as pagers, fax machines, and electronic mail services will be as easy to reach as the local printer.

If you imagine the operating system growing by leaps and bounds, fear not. Dynamically linked libraries permit users to maintain trimmer operating systems. For example, you don't have to extend the system with any library file that provides unnecessary services. If support of sending messages to pagers is not needed, remove that extension file from the appropriate folder (or directory)—or simply avoid buying unneeded extensions.

Microsoft and Apple have already demonstrated the use of shared and dynamically linked libraries to extend the operating system with new features, such as Apple's support of QuickTime, a movie/soundtrack type of data. Microsoft has used dynamic linking to unobtrusively add pen extensions to Windows (see "Windows Meets the Pen" in the June '92 issue of *Byte Magazine*). It even allows applications that are not pen-aware to respond to pen input. (The library is dynamically linked to the operating system's input drivers, enabling pen inputs to be intercepted and processed into equivalent mouse or keystroke inputs.)

Dynamic linking theoretically supports ad hoc construc-

tion of applications by users. This will offer users the exciting prospect of being able to use modules from different vendors cooperatively—as if they were one application. Imagine taking your favorite spreadsheet, word processor, and page-layout programs and developing compound documents where the spreadsheets are maintained by the spreadsheet, stories are maintained by the word processor, and the page appearance by the page-layout program. You should even be able to save those modules as a uniquely named application that runs all the modules when launched—as though they were one application.

I know I am not the only one who sees these coming attractions. However, data-interchange standards need to mature further to allow increased inter-application communication. Nevertheless, companies such as Apple are pro-

(Continued on page 42.)

Press Watch

Hurrah! A couple of Forth-related stories have recently appeared in *Midnight Engineering* (March/April issue) and in *Dr. Dobbs* (June issue). By the time you are able to read this, a third article focusing upon Forth will have appeared in the weekly publication *EE Times* (see the July 6 issue).

For those of you who did not see the articles recently published, here are a few of the highlights. *Midnight Engineering* portrayed Charles Moore as an engineer-entrepreneur. Their article retraces his steps beginning with the creation of Forth. The article lingers on the topic of Chuck's newer Forth chips and the exciting chip-design software with which he is developing a new generation of microprocessors.

Dr. Dobbs ran an article by Jack Woehr in which he recreates a fascinating dinner conversation between many of the ANS Forth committee members and Charles Moore. Besides Jack Woehr and Charles Moore, the cast features Ray Valdez (an editor at the magazine), Elizabeth Rather, George Shaw, John Stevenson, and Mitch Bradley. The far-ranging discussion lingers on Chuck's work on new microprocessors, and the software he has created to help design and simulate his newest microprocessor. There is also a short sidebar about ANS Forth.

(As recently as a few years ago, *Dr. Dobbs* published a regular Forth column by Martin Tracy. More recently, *Embedded Systems Journal* ran a series of columns by Jack Woehr.)

As ANS Forth is nearing completion, it's possible to obtain more Forth coverage in these and other journals. Take heart Forth authors! Let's tell more of the stories that remain Forth folk tales. Stories that need telling include the use of Forth in space shuttle experiments, the real-time programming contest that introduced the much-admired "gizmo," the open-boot ROM, Forth simulations of genetic evolution processes, Forth-programmed devices that help the physically handicapped, the popularity of Forth amongst European scientists and programmers, and so forth.

Product Watch

May 1992

AM Research announced the amr451, an integrated microcontroller development system that uses a PC host for Forth compilation and interpretation. It targets the associated arm451 Application System, which has attractive quantity prices. Besides including the Signetics SC80C451 (which runs 8031, 8051, and 8751 code), both systems have options for ROM and RAM, RS-485, RS-232, LCD, keypad, prototyping daughter boards, and A/D converters. To be able to emulate ROM and break the umbilical connection to the host, you use TURNKEY to write an autostart vector to battery-backed RAM in the development system. AM Research also provides manufacturing and testing services for products you prototype.

June 1992

Vesta Technology announced a standalone, 18-bit A/D converter on a business-card-size PC that communicates through its own RS-232 communications port. The RS-232 line can optionally serve as the physical connection between up to 32 such (data collecting) units and a master (listening and polling) node, such as a PC or one of Vesta's own single-board computers. A downloadable manual for this product is available on the Vesta BBS at 303-278-0364.

July 1992

Silicon Composers announced a 12 MIPS, 32-bit data acquisition system, the DAS32. This system bundles a daughter board well-equipped with I/O ports (including SCSI) and Forth development software with either one of two single-board computers based on the SC32 Forth microprocessor—the SBC32 or the PCS32. The DAS32-SA is based on the SBC32 computer, which can communicate with a PC or terminal over a serial line. The DAS32-PC is based on the PCS32 computer, which plugs directly into a PC slot.

Companies Mentioned

AM Research
4600 Hidden Oaks Lane
Loomis, California 95650
Phone: 916-652-7472

Silicon Composers
208 California Ave.
Palo Alto, California 94306
Phone: 415-322-8763

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

News from the Forth RoundTable

Best of GEnie

Gary Smith
Little Rock, Arkansas

News from the GEnie Forth RoundTable—This issue's column again comes from Category 10, Topic 30 "What should the Standard include?" where the focus has shifted to what actually constitutes system-/model-specific versus implementation-dependent. If you thought the ANS Forth standard was all but set, you were mistaken. This discussion helps disclose some of the mechanics that make a software standard so difficult to impose on differing hardware and processor platforms.

Before I share the exchanges with you, I need to take a moment to thank Bob Lee for stepping in to continue ForthNet ports from Usenet after personal problems forced me to vacate my position as a Forth RoundTable SysOp and to abdicate my role as the primary ForthNet bridge to GEnie. As many of you know, I was instrumental in the creation of ForthNet and this made stepping aside even more difficult. Bob, through his generous offer to continue the Usenet-to-GEnie bridge via his Citadel site, made a painful decision more palatable. Thanks, Bob.

Those who still wish to reach me, please note my new signature file:
Gary Smith, P. O. Drawer 7680
Little Rock, Arkansas 72217 U.S.A.
uunet!ddi1!lrark!glsrk!gars
nuucp%ddi1@uunet.UU.NET
GEnie Unix SysOp (GARY-S), phone: 501-228-5182
fax:501-228-9374 (0800–1700 GMT-6)

We might be better off if long-time vendors tried harder to make their favorite ideas and best-selling points standard practice.

What Should the Standard Include?

Implementation Model
From Nick_Janow @ mindlink.bc.ca
British Columbia, Canada

Several people have been saying that dpANS-3 is the wrong approach and that ANSI Forth should be built on a standard implementation model (or virtual machine). This standard implementation model would allow a concise definition of Forth, common debugging tools, and other benefits.

While these benefits are valid, the advocates don't mention the problems with that approach. It limits Forth to one implementation. It would end experimentation with vocabulary systems, threading techniques, etc. It would also break existing code. It would limit the hardware the compilers could run efficiently on.

I like the concept of a Forth based on a standard implementation model, and I think a lot of Forthers do, too. The benefits are valid. However, I expect that each of the programmers who would like a standard implementation would want *their* particular implementation to be the standard one. One person wants linked lists, another wants hashed lists; one wants the simplicity of ITD, another wants the speed of JSR threading. How do we choose the "standard implementation"?

I'm open to arguments that will prove that the present dpANS would be worse for Forth than an implementation standard (taking Forth vendors, commercial developers, and all other important groups into consideration). The arguments would also have to show that it would be possible to get agreement on a particular implementation model.

Is anyone willing to propose an implementation standard (or even part of one) along with arguments as to why it should be chosen over the other alternatives?

Re: Implementation Model
From mikc @ hal.gnu.ai.mit.edu (/etc/organization)

In article <12990@mindlink.bc.ca>, Nick Janow writes:

> Several people have been saying that dpANS-3 is the wrong approach and that ANSI Forth should be built on a standard implementation model (or virtual machine). This standard implementation model would allow a concise definition of Forth, common debugging tools, and other benefits.
>
> While these benefits are valid, the advocates don't mention the problems with that approach. It limits Forth to one implementation. It would end experimentation with vocabulary systems, threading techniques, etc. It would also break existing code. It would limit the hardware the compilers could run efficiently on.

We've been discussing the problems with that approach constantly. Less has been said about how the approach taken by the TC *[ANS Technical Committee]* would solve these problems. Standard portable Forth code is very restrictive. If there was a large body of code that we all insisted had to run on every Forth system, that would constrain Forth as much as having a standard Forth model implementation. Saying that Basis allows us to write portable code but does not stop us from experimenting with new techniques makes no sense to me.

> I like the concept of a Forth based on a standard implementation model, and I think a lot of Forthers do, too. The benefits are valid. However, I expect that each of the programmers who would like a standard implementation would want *their* particular implementation to be the standard one. One person wants linked lists, another wants hashed lists; one wants the simplicity of ITD, another wants the speed of JSR threading. How do we choose the "standard implementation"?

Some capable experienced Forth programmers don't try

to make their particular implementation the standard one. There is a feeling of possesiveness about Forth systems; good ideas are kept for the benefit of their author's business and to be sold to their customers. If a vendor does make an attempt to present their particular way of doing things, it meets with dissaproval from others who do not want to change their way of doing things. We might be better off if the long-time vendors tried harder to make their favorite ideas and best-selling points standard practice.

> I'm open to arguments that will prove that the present dpANS would be worse for Forth than an implementation standard (taking Forth vendors, commercial developers, and all other important groups into consideration). The arguments would also have to show that it would be possible to get agreement on a particular implementation model.

The proof is the complaint from those who have large amounts of code to maintain that adopting dpANS would put them out of business. Adopting a particular implementation model would have a similar bad effect. The standards committee has spent over four years debating to reach its decisions. But this debate was between a relatively few people. Every Forth programmer has to make these same decisions and reach the same conclusions if one person's Forth code is to look like another person's Forth code. So I don't think a committee can do much to change or "modernize" Forth. It should stick to documenting old Forth as it was years ago. Some incompatible Forth practices might be resolved by a standards committee, but most of these were introduced by past standards committees.

To those most interested in making Forth "up-to-date," I would like to ask—how do you propose to retrain old Forth programmers in your new methods? The ANSI standard document is a terrible way to do this. I have been clearly told that a standard document is not a tutorial. (This is one reason why standard documents are so badly written.) How are you going to communicate the reasons for your new decisions? If a standards document is written only for those experienced in the subject and is not a tutorial, then it can only refer to materal already widely known, and can't intoduce anything new.

But I believe a standards document is a tutorial and it should introduce new ideas. Portable Forth code is a new idea. The document then has to be inspected and tested as a tutorial. This has not been done. Another test that should have been made is to ask—Are these new words such an improvement that it is worth changing old code? Will customers stop using their old systems and change to the new ANS-compliant ones? Perhaps that test is too difficult for anything to pass.

Now suppose a new implementation of a modern up-to-date Forth system was written. Naturally it would have clear documentation, a tutorial, and complete well-commented source code. It would be given to students to learn on their own from the material presented. It would be used in industry when a portable Forth was needed for new projects. Eventually it would become so well known that programmers would change the code written on older systems.

I know there will be great difficulties in producing a better standards document or a portable Forth implementation. But doesn't a model implementation have as much chance in improving Forth as an ANSI standard document?

> Is anyone willing to propose an implementation standard (or even part of one) along with arguments as to why it should be chosen over the other alternatives?

There have been and will be several people who will write sample ANSI implementations. It will be much harder to write the arguments as to why these should be chosen over other alternatives.

I hope the above will illustrate my concern that we have too many completely different tasks wrapped up in the standards effort. An ANSI standard that was just a rehash of *Starting Forth* would be perfectly fine with me, but trying to combine a big list of revised Forth words with the idea that this is going to show us how to step into the future is not my idea of what should be done.

"Trash ANS, trash C (and sense, as well)"

From dak @ kaa.informatik.rwth
(Rechnerbetrieb Informatik / RWTH Aachen)

C is a language which should never have become standardized. There is simply no portable way in C to write a decompiler for any machine. Also, it is not possible with today's C standards to implement debugging facilities which are portable (at least, not if they have to include disassembly and use the debug features of the processor). Pascal should never have been standardized as well. Instead one should have prescribed a system such as the UCSD p-code as a standard, together with all the binaries. That way disassemblers, decompilers, debuggers would have been portable. Who cares about a speed factor of five? Who cares about improvement or bug fixes?

Those in for performance: we should have standardized assembly languages long ago. Why develop processors capable of anything new?

And dear ANS: please standardize Forth in a way that leaves open only one way of implementing anything. Prescribe a machine language. Prescribe word sizes. Prescribe 'hidden features' (a.k.a. bugs & nuisances). Prescribe threading. Prescribe everything which might be implemented differently in a more efficient way in the future. Establish Forth as dead and inflexible, not to be implemented on any new processor, and as an especially inefficient language.

Follow the market leader Intel, whose high-performance chips (P5 to come) are source code compatible with the 8080, one of the earliest 8-bit processors. Ruin your opcodes! Hamper your registers! Throw away chip space! Waste memory and performance! Prohibit accessing more memory or performance than you could afford in your youth!

Don't try to describe and preserve the flavor of Forth! Concentrate on its aftertaste! Its indigestiveness! Its nuisances! The color of vomited binaries! Come on!

From B.RODRIGUEZ2 [Brad]
bradford@maccs.dcss.mcmaster.ca
Brian:

Briefly, all of my applications used some form of "lazy evaluation." I'm not the only one using this; there was a good article in *The Computer Journal* #43 (March/April 1990) by Marla Bartel. One of my examples is the word

```
: &   ( f -- f)   ( t -- )
  0= IF 0 R> DROP THEN ;
```

which if entered with "false," forces a word to exit with false on the stack, but if entered with "true," simply consumes the flag and proceeds. This turns out to be a surprisingly useful and powerful construct, as you can see from the references cited in my previous posting. The only way to do this in dpANS Forth is:

```
: &   POSTPONE 0= POSTPONE
  IF POSTPONE FALSE POSTPONE EXIT
  POSTPONE THEN ; IMMEDIATE
```

which will typically compile five cells rather than one... and I use this construct a *lot*.

I decided to post my previous comment after the discussion turned around to stack addressibility. At first I thought I might need an addressible stack, but upon reflection I realized that all I need was the entitlement to do R> DROP, i.e., some recognition that when a high-level definition is entered, the top of the return stack contains a single-cell return address.

I seem to be in a position similar to Dr. Wavrik: here's a technique I've used for years—and, I should add, which works in all of the twenty-odd Forths I've used—which is being lost in ANS Forth. I have no hope of reinstating this entitlement. (Sigh.)

"I've used MS-DOS; I can use ANS Forth. But not for this."

From B.DUNN5 [Brian]

B.RODRIGUEZ2 [Brad] writes:

> ```
> : &   ( f -- f)   ( t -- )
>   0= IF 0 R> DROP THEN ;
> ```
> which if entered with "false," forces a word to exit with false on the stack, but if entered with "true," simply consumes the flag and proceeds.
>
> The only way to do this in dpANS Forth is
> ```
> : &   POSTPONE 0= POSTPONE
>   IF POSTPONE FALSE POSTPONE EXIT
>   POSTPONE THEN ; IMMEDIATE
> ```
> which will typically compile five cells rather than one... and I use this construct a *lot*.

I see a word called UNLOOP which does almost what you want, except for do loops. How about proposing a word UNCALL or UNNEST? If they saw fit to include UNLOOP, I don't see how they can't include UNCALL.

And if they don't, I might as well mention the protostandard category on GEnie, which might one day look like an 'ANS fixup category.'

But this is just a patch for one little problem, which is a symptom of a larger problem, about which you have a good point. If it were assumed that a call placed a value on the return stack, and a do loop placed three, you could get rid of UNLOOP and UNCALL altogether (although I like them anyhow).

From B.RODRIGUEZ2 [Brad]
bradford@maccs.dcss.mcmaster.ca
Brian:

UNCALL might work, except for my pattern-matching package, which is positively ruthless about shuttling values—including return addresses—back and forth between the two stacks.

I'm not the only one playing such games. I've read a paper by M.L. Gassanenko of the University of Leningrad describing his BacFORTH which solves the same problem in a totally different way—which also requires carnal knowledge of the return stack. Gassanenko's paper seems to me to be among the leading edge of Forth language research; it's a pity that his innovative ideas are about to be torpedoed.

By the way, you can't assume that DO puts three values on the stack. Some implementations only put two. And I've been told that the dpANS leaves open the option of an independent loop stack (á la STOIC).

I'm aware of the protostandard category on GEnie; I've been monitoring it, although I haven't had time to post to it yet. I fear that you are right: already I'm hearing of TC members looking forward to the *next* ANS Forth as the "fixed-up" standard. Not a real vote of confidence in the current dpANS.

"I've used MS-DOS; I can use ANS Forth."

Re: What should the Standard include?
From dwp @ willett.pgh.pa.us

In article
<9206191017.AA11267@nettlerash.berkeley.edu>,
Anton Martin Ertl writes:

> ANSI Forth's purpose (more correctly, X3J14's purpose) is (among other conflicting things) to standardize existing practice, not create a new language or add things never tried before.
>
> Writing decompilers, debuggers, etc. in Forth is existing practice. So if you don't want this capability in ANSI Forth, use a different argument.

ANS Forth is not aiming at the portability of the Forth system itself, but the portable *use* of such a system. The debuggers, decompilers, etc. are inherently a part of the Forth system because they depend on system-specific details (threading vs. native code, access to return stack and IP, and on and on...). Support for those kinds of tools would require either a model-based standard, or an Abstract Data Type interface to the facilities needed. Good or not (I personally think good), a model-based standard is not going to happen, at least not this time around. The Internals mailing list is working on the ADT approach, but it is not at a standardizable point yet.

> I am not asking for something that I don't know what it is. I point out that ANSI Forth does not standardize capabilities that are present in all Forth systems and that could be portable (i.e., they are not processor- or OS-dependent.

Addressable stacks are processor-dependent (i.e., *not* present in all Forth systems and *not* portable). The structure of the Forth dictionary is implemetation-dependent. The implementation will depend on the underlying hardware, therefore the structure of the dictionary is indirectly processor-dependent. I'm very curious to see what the group working on internals structure has to say about this, when they reach a consensus. Again, such a consensus does not already exist and therefore cannot be standardized.

> Addressing the stack has been called bad programming style and I agree. But that's no reason to throw it out.

Especially when there are much better reasons. Addressable stacks are processor-dependent. Requiring them to be addressable incurs substantial penalties on some processors, and would require a strong argument to do so. I have seen no argument for addressable stacks, other than some hypothetical potentiality.

> The disagreement is where to draw the line between language and implementation. In my opinion dpANS Forth makes the language part very small, making many Forth programming techniques implementation-dependent and thus non-portable.

I think it would help to be more precise here. There is a difference between *not* saying that something *is* portable, and making something non-portable. It seems to me that the diversity of Forth implementation strategies indicates that there are many who are not satisfied with the old fig-Model "traditional Forth" type approach. Just look at what Forth's creator has done with the language. If we admit that he was a visionary in creating Forth (not in one fell swoop, but with an evolutionary process), then either he has fallen from the/his way by not sticking with the same model, or he understands the essence of Forth as not being tied to a single model. Perhaps he is just a non-conformist who accidently struck upon a good idea, but since has wandered off to other things?

It seems to me to be unrealistically idealistic to expect a popular return to a model-based de facto/ANSI standard. Forth is too individualistic for that. Too many people have already made their own evaluations of the tradeoffs in traditional Forth and found other tradeoffs to be more productive, useful, helpful, etc.

I would very much like to hear what those who are advocating a model-based approach (and a "traditional" model at that) to the ANS standard think about *why* there has been so much diversity to Forth implementations and why so many have chosen alternatives to "traditional" Forth.

Re: What should the Standard include?
From dwp @ willett.pgh.pa.us

In article <3789.UUL1.3#5129@willett.pgh.pa.us>, Brad Rodriguez writes:

> I'm glad to see it's not too late for me to post *my* examples of what I can't do in ANS Forth. I have
> a) a parser [sigForth newsletter vol.2 no.2],
> b) a pattern matching extension [FORML '89 proceedings], and
> c) a fast expert system [Rochester '90 proceedings],
>
> all of which are reasonably portable, and all of which have been invalidated by ANS Forth. The latter is particularly significant to me, as I am using it in my University research.

For those of us without access to those publications, could you relate the essence of what it is you do that has been invalidated by dpANS Forth?

Re: What should the Standard include?
From anton @ mips.complang.tuwie
(Institut fuer Computersprachen, Technische Universitaet Wien)

In article <3815.UUL1.3#5129@willett.pgh.pa.us>, Doug Philips writes:

> In article <9206191017.AA11267@nettlerash.berkeley.edu>, Anton Martin Ertl writes:
>
> Writing decompilers, debuggers, etc. in Forth is existing practice.
>
> ANS Forth is not aiming at the portability of the Forth system itself, but the portable *use* of such a system. The debuggers, decompilers, etc. are inherently a part of the Forth system because they depend on system-specific details (threading vs. native code, access to return stack and IP, and on and on...).

Who says what's system-specific (a.k.a. implementation-dependent) and what's not? Hmm, it's the standard: What the standard defines is not system-specific, and what it does not define is system-specific. Therefore: If we add ways for implementing debuggers, decompilers, etc. to the standard (e.g., in an ADT approach or by specifying a standard model), they are no longer system-specific.

> I am not asking for something that I don't know what it is. I point out that ANSI Forth does not standardize capabilities that are present in all Forth systems and that could be portable (i.e., they are not processor- or OS-dependent.
>
> Addressable stacks are processor-dependent (i.e., *not* present in all Forth systems and *not* portable).

Implementation-dependent means "not present in all Forth systems." Processor-dependent means not implementable on all processors (e.g., memory > 64K).

> The structure of the Forth dictionary is implementation-dependent. The implementation will depend on the underlying hardware, therefore the structure of the dictionary is indirectly processor-dependent.

You are saying that there are processors that cannot process linked lists (or hash tables or an ADT for the dictionary). Don't make me laugh so hard!

> The disagreement is where to draw the line between language

> and implementation. In my opinion dpANS Forth makes the language part very small, making many Forth programming techniques implementation-dependent and thus non-portable.
>
> I think it would help to be more precise here. There is a difference between *not* saying that something *is* portable, and making something non-portable.

There may be a difference, but it does not matter. If nothing else happens, anything not covered by the standard will be non-portable.

> I would very much like to hear what those who are advocating a model-based approach (and a "traditional" model at that) to the ANS standard think about *why* there has been so much diversity to Forth implementations and why so many have chosen alternatives to "traditional" Forth.

Well, Forth is easy to implement, so many people do it. And they won't just type in fig-Forth listings, they experiment. They add new features, they make it faster, they leave out what's inessential (to them), make it compliant to a new standard document, etc. (Of course, there are also implementations not derived from fig-Forth.) Some of those implementations are distributed. Then other people will use these implementations. Some will know about the changes (and won't care), and some will not know about them and may have a hard awakening.

"Some things have to be seen to be believed.
Most things have to be believed to be seen."

Re: What should the Standard include?
From dwp @ willett.pgh.pa.us

In article <3830.UUL1.3#5129@willett.pgh.pa.us>, Brad Rodriguez writes:

> Brian: UNCALL might work, except for my pattern-matching package, which is positively ruthless about shuttling values—including return addresses—back and forth between the two stacks.

So your & won't work in those cases either. Perhaps it would be helpful to know what kinds of things you are doing with return addresses. Are you just moving them around? Are you modifying them?

Some systems have words beyond just R>, >R, and R@. Perhaps what is needed is a portable way to manipulate return stack "items" which are not data pushed from the data stack for temporary space. Would having a word, say R-CELLS, that gives you the number of cells in a return stack "address/item" be enough? How many of the data stack words (DUP, SWAP, ROLL, etc.) have you written return stack equivalents for?

> I'm not the only one playing such games. I've read a paper by M.L. Gassanenko of the University of Leningrad describing his BacFORTH which solves the same problem in a totally different way—which also requires carnal knowledge of the return stack. Gassanenko's paper seems to me to be among the leading edge of Forth language research; it's a pity that his innovative ideas are about to be torpedoed.

Really? Is ANSI going to take away the systems he is using now? Are they going to sabotage/cancel FORML conferences? Please be more specific as to how you think leaving carnal knowledge out of the dpANS will kill innovation.

> By the way, you can't assume that DO puts three values on the stack. Some implementations only put two. And I've been told that the dpANS leaves open the option of an independent loop stack (á la STOIC).

I would much rather, from a philosophic point of view, that they had required a completely independent loop stack. When I first learned Forth I couldn't believe the caveat about accessing the return stack across a DO LOOP boundary... talk about a bad joke! Of course, a separate DO LOOP stack won't break existing code, as it will only allow things that weren't allowed before, and not visa versa... but I can see why they couldn't require it.

Re: What should the Standard include?
From dwp @ willett.pgh.pa.us

In article <1992Jul1.075122.19668@email.tuwien.ac.at>, Anton Martin Ertl writes:

> In article <3815.UUL1.3#5129@willett.pgh.pa.us>, Doug Philips writes:
>
> ANS Forth is not aiming at the portability of the Forth system itself, but the portable *use* of such a system. The debuggers, decompilers, etc. are inherently a part of the Forth system because they depend on system-specific details (threading vs. native code, access to return stack and IP, and on and on...).
>
> Who says what's system-specific (a.k.a. implementation-dependent) and what's not?

I've already asked "Who gets to say what Forth is?" The answer to your question is not unrelated to the answer to that question.

So far I haven't seen any answer, except for the "I do" implicit in every post that says what Forth is (traditional or not).

> Hmm, it's the standard: What the standard defines is not system-specific, and what it does not define is system-specific.

What the standard defines is not system-specific, and what it does not define it does not define. That doesn't mean that additional portability is impossible, or even disallowed, it just isn't required for the ANSI "stamp."

> Therefore: If we add ways for implementing debuggers, decompilers, etc. to the standard (e.g., in an ADT approach or by specifying a standard model), they are no longer system-specific.

Of course. If the internals work now under way had begun five years ago, and had it been actually implemented in more than a handful of systems, then perhaps it could have made it into this standard. As it stands, it won't.

(Continued on page 42.)

Forth resources & contact information

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

Board of Directors
John Hall, President
Jack Woehr, Vice-President
Mike Elola, Secretary
Dennis Ruffer, Treasurer
David Petty
Nicholas Solntseff
C.H. Ting

Founding Directors
William Ragsdale
Kim Harris
Dave Boulton
Dave Kilbridge
John James

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.

On-Line Resources

ForthNet

ForthNet is a virtual Forth network that links designated message bases of several bulletin boards and information services in an attempt to provide greater distribution of Forth-related info.

ForthNet is provided courtesy of the SysOps of its various links, who shunt appropriate messages in a manual or semi-manual manner. The current branches of ForthNet include UseNet's comp.lang.forth, BitNet's FIGI-L, the bulletin board systems RCFB, ACFB, LMI BBS, Grapevine, and FIG's RoundTable on GEnie. (Information on modem-accessible systems is included below.)

The various branches of ForthNet do not have the same rules of appropriate postings or etiquette. Many bulletin board posts are very chatty and contain some personal information, and some also contain blatant commercial advertising. Most comp.lang.forth posts are not like that. ForthNet messages that are ported into comp.lang.forth from the rest of the ForthNet all originate on GEnie, which is a kind of *de facto* ForthNet message hub. All such messages are ported to comp.lang.forth with a from-line of the form: From: ForthNet@willett.pgh.pa.us ...

Most messages ported to comp.lang.forth also contain some trailer information as to where they actually originated, if it was not on GEnie.

There is no e-mail link between the various branches of ForthNet. If you need to get a message through to someone on another branch, please either make your message general enough to be of interest to the whole net, or contact said person by phone, U.S. Mail, or some other means. Thoughtful message authors place a few lines at the end of their messages describing how to contact them (electronically or otherwise).

Phone information for the dial-in services mentioned above:

RCFB (Real-Time Control Forth Board) 303-278-0364
SysOp: Jack Woehr SprintNet node coden
Location: Denver, Colorado, USA

ACFB
(Australia Connection Forth Board) 03-809-1787 in Australia
SysOp: Lance Collins 61-3-809-1787 International
Location: Melbourne, Victoria, AUSTRALIA

LMI BBS (Laboratory Microsystems, Inc.) 213-306-3530
SysOp: Ray Duncan SprintNet node calan
Location: Marina del Ray, California, USA

Grapevine (Grapevine RIME hub) 501-753-8121 to register
SysOp: Jim Wenzel 501-753-6859 thereafter
Location: Little Rock, Arkansas, USA

GEnie (General Electric Network for
Information Service) 800-638-9636 for info.
SysOps: Dennis Ruffer (D.RUFFER)
Leonard Morgenstern (NMORGENSTERN)
Gary Smith (GARY-S)
Location: Forth RoundTable—type M710 or FORTH

Forth Libraries

There are several repositories of Forth programs, sources, executables, and so on. These various repositories are *not* identical copies of the same things. Material is available on an *as-is* basis due to the charity of the people involved in maintaining the libraries. There are several ways to access Forth libraries:

FTP

Note: You can only use FTP if you are on an Internet site which supports FTP (some sites may restrict certain classes of users). If you have any questions about this, contact your system administrator for information. Your system administrator should always be your first resort if you have any difficulties or questions about using FTP.

For MS-DOS-related files, there are currently two sites from which you can anonymously FTP Forth-related materials:
WSMR-SIMTEL20.ARMY.MIL (Simtel20 for short)
WUARCHIVE.WUSTL.EDU (Wuarchive for short)

Wuarchive maintains a "mirror" of the material available on Simtel20. Simtel20 has a limited amount of material, most of it binaries for MS-DOS computers. The Forth files on Simtel20 are in directory PD1:<MSDOS.FORTH>. The Forth files on Wuarchive are in directory /mirror/msdos/forth. For detailed information on how use FTP and the Simtel20 archive (it is too much to include here), see the text files in:
PD1:<MSDOS.STARTER>SIMTEL20.INF *or*
/mirrors/starter/simtel20.inf

An FTP site containing a mirror of the FIG library on GEnie is "under construction" and will be announced when it is ready.

FIGI-L Gateway

For those who have access to BITNET/CSNet but not Usenet, comp.lang.forth is echoed in FIGI-L. The maintainer of the Internet/BITNET gateway since first quarter 1992 is as follows:

Pedro Luis Prospero Sanchez internet: pl@lsi.usp.br (PREFERRED)
University of Sao Paulo uunet: uunet!vme131!pl
Dept. of Electronic Engineering hepnet: psanchez@uspif1.hepnet
phone: (055)(11)211-4574
home: (055)(11)914-9756
fax: (055)(11)815-4272

Modem

For those desiring to use (or stuck with) modems, the dial-in systems listed above also have Forth libraries.

Note: If you are unable to access SIMTEL20 via Internet FTP or through one of the BITNET/EARN file servers, most SIMTEL20 MS-DOS files, including the PC- network at 313-885-3956. DDC has multiple lines which support 300/1200/2400/9600/14400 bps (HST/V.32/V.42/V.42bis/MNP5). This is a subscription system with an average hourly cost of 17 cents. It is also accessible on Telenet via PC Pursuit, and on Tymnet via StarLink outdial. New files uploaded to SIMTEL20 are usually available on DDC within 24 hours.

Information provided by:
Keith Petersen Maintainer of SIMTEL20's MSDOS,
MISC & CP/M archives [IP address 26.2.0.74]
Internet: w8sdz@WSMR-SIMTEL20.Army.Mil or
w8sdz@vela.acs.oakland.edu
Uucp: uunet!wsmr-simtel20.army.mil!w8sdz
BITNET: w8sdz@OAKLAND

This list was compiled 20 February 1992. While every attempt was made to produce an accurate list, errors are always possible. Sites are also subject to mechanical problems or SysOp burnout. Please report any discrepancies, additions, or deletions to the following:

Gary Smith
P. O. Drawer 7680
Little Rock,AR 72217
U.S.A.

uunet!ddi1!lrark!glsrk!gars
nuucp%ddi1@uunet.UU.NET
GEnie Forth RT & Unix RT SysOp
phone: 501-228-5182
fax: 501-228-9374
(0800–1700 GMT-6)

E-Mail

For those with e-mail-only access, there is not much. For now, posts from ForthNet ported into comp.lang.forth sometimes advertise files being available on GEnie. Those messages also contain information on how to get UU encoded e-mail copies of the same files. There is an automated e-mail service. The entire FIG library on GEnie is available via e-mail, but no master index or catalog is yet available. The file FILES.ARC contains a fairly recent list of the files on GEnie, and files added since then are only documented for comp.lang.forth readers by way of the "Files On-line" messages ported through ForthNet.

If you have any questions about ForthNet/comp.lang.forth or any information to add/delete or correct in this message, or any suggestions on formatting or presentation, please contact either Doug Philips or Gary Smith (preferably both, but one is okay) via the following addresses:

- Internet: dwp@willett.pgh.pa.us
  or ddi1!lrark!gars@uunet.uu.net
- Usenet: ...!uunet!ddi1!lrark!gars
  or ...!uunet!willett.pgh.pa.us!dwp
- GEnie: GARY-S or D.PHILIPS3
- ForthNet: Grapevine, Gary Smith
  leave mail in Main Conference (0)

To communicate with the following, set your modem and communication software to 300/1200/2400 baud with eight bits, no parity, and one stop bit, unless noted otherwise. GEnie requires local echo (half duplex).

GEnie*

For information, call 800-638-9636

- Forth RoundTable (ForthNet*)
  Call GEnie local node, then
  type M710 or FORTH
  SysOps:
  Dennis Ruffer (D.RUFFER)
  Leonard Morgenstern (NMORGENSTERN)
  Gary Smith (GARY-S)
  Elliott Chapin (ELLIOTT.C)

BIX (Byte Information eXchange)

For Information, call 800-227-2983

- Forth Conference
  Access BIX via TymNet, then type j forth
  Type FORTH at the : prompt
  SysOp: Phil Wasson

CompuServe

For Information, call 800-848-8990

- Creative Solutions Conf.
  Type !Go FORTH
  SysOps: Don Colburn, Zach Zacharia, Ward McFarland, Greg Guerin, John Baxter, John Jeppson

- Computer Language Magazine
  Type ! Go CLM
  SysOps: Jim Kyle, Jeff Brenton, Chip Rabinowitz, Regina Star Ridley

The WELL (Unix BBS with PicoSpan frontend)

- Forth conference
  Access WELL via CPN (CompuServe Packet Net)
  or via SprintNet node: casfa
  or 415-332-6106
  Forth Fairwitness: Jack Woehr (jax)
  Type ! j forth

Citadel Network - two sites

- Undermind (UseNet/Citadel bridge)
  Atlanta, GA
  404-521-0445

**GEnie is the repository of the Forth Interest Group's official Forth Library.*

- Interface (formerly Nite Owl)
  SysOp: Bob Lee
  Napa, CA
  707-823-3052

Non-Forth-specific BBS's with extensive Forth libraries:

- DataBit
  Alexandria, VA
  703-719-9648
  SprintNet node dcwas

- Programmer's Corner
  Baltimore/Columbia, MD
  301-596-1180 or
  301-995-3744
  SprintNet node dcwas

- PDS*SIG
  San Jose, CA
  408-270-0250
  SprintNet node casjo

International Forth BBS's

See Melbourne Australia in ForthNet node list above

- Serveur Forth
  Paris, France
  From Germany add prefix 0033
  From other countries add 33
  (1) 41 08 11 75
  300 baud (8N1) *or*
  1200/75 E71 *or*
  (1) 41 08 11 11
  1200 to 9600 baud (8N1)
  For details about high-speed,
  Minitel, or alternate carrier
  contact: SysOp Marc Petremann
  17 rue de la Lancette
  Paris, France F-75012

- SweFIG
  Per Alm Sweden
  46-8-71-35751

- NEXUS Servicios de Informacion, S.L.
  Travesera de Dalt, 104-106
  Entlo. 4-5
  08024 Barcelona, Spain
  + 34 3 2103355 (voice)
  + 34 3 2147262 (data)

- Max BBS (ForthNet*)
  United Kingdom
  0905 754157
  SysOp: Jon Brooks

- Sky Port (ForthNet*)
  United Kingdom
  44-1-294-1006
  SysOp: Andy Brimson

- Art of Programming
  Mission, British Columbia, Canada
  604-826-9663
  SysOp: Kenneth O'Heskin

- The Forth Board
  Vancouver, British Columbia, Canada
  604-681 3257
  Forth-BC Computer Society

U'NI-net/US

- The Monument Board (U'NI-net/RIME ForthNet bridge)
  Monument, CO
  Jerry Shifrin (ForthNet charter founder)
  719-488-9470

FIG Chapters

The Forth Interest Group Chapters listed below are currently registered as active with regular meetings. If your chapter listing is missing or incorrect, please contact the FIG office's Chapter Desk. This listing will be updated regularly in Forth Dimensions. *If you would like to begin a FIG Chapter in your area, write for a "Chapter Kit and Application."*

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

U.S.A.

- **ALABAMA**
  Huntsville Chapter
  Tom Konantz
  (205) 881-6483

- **ALASKA**
  Kodiak Area Chapter
  Ric Shepard
  Box 1344
  Kodiak, Alaska 99615

- **ARIZONA**
  Phoenix Chapter
  4th Thurs., 7:30 p.m.
  Arizona State Univ.
  Memorial Union, 2nd floor
  Dennis L. Wilson
  (602) 381-1146

- **CALIFORNIA**
  Los Angeles Chapter
  4th Sat., 10 a.m.
  Hawthorne Public Library
  12700 S. Grevillea Ave.
  Phillip Wasson
  (213) 649-1428

  North Bay Chapter
  2nd Sat.
  12 noon tutorial, 1 p.m. Forth
  2055 Center St., Berkeley
  Leonard Morgenstern
  (415) 376-5241

  Orange County Chapter
  4th Wed., 7 p.m.
  Fullerton Savings
  Huntington Beach
  Noshir Jesung (714) 842-3032

  Sacramento Chapter
  4th Wed., 7 p.m.
  1708-59th St., Room A
  Bob Nash
  (916) 487-2044

  San Diego Chapter
  Thursdays, 12 Noon
  Guy Kelly (619) 454-1307

  Silicon Valley Chapter
  4th Sat., 10 a.m.
  Applied Bio Systems
  Foster City
  John Hall
  (415) 535-1294

  Stockton Chapter
  Doug Dillon (209) 931-2448

- **COLORADO**
  Denver Chapter
  1st Mon., 7 p.m.
  Clifford King (303) 693-3413

- **FLORIDA**
  Orlando Chapter
  Every other Wed., 8 p.m.
  Herman B. Gibson
  (305) 855-4790

- **GEORGIA**
  Atlanta Chapter
  3rd Tues., 7 p.m.
  Emprise Corp., Marietta
  Don Schrader (404) 428-0811

- **ILLINOIS**
  Cache Forth Chapter
  Oak Park
  Clyde W. Phillips, Jr.
  (708) 713-5365

  Central Illinois Chapter
  Champaign
  Robert Illyes (217) 359-6039

- **INDIANA**
  Fort Wayne Chapter
  2nd Tues., 7 p.m.
  I/P Univ. Campus
  B71 Neff Hall
  Blair MacDermid
  (219) 749-2042

- **IOWA**
  Central Iowa FIG Chapter
  1st Tues., 7:30 p.m.
  Iowa State Univ.
  214 Comp. Sci.
  Rodrick Eldridge
  (515) 294-5659

  Fairfield FIG Chapter
  4th Day, 8:15 p.m.
  Gurdy Leete (515) 472-7782

- **MARYLAND**
  MDFIG
  3rd Wed., 6:30 p.m.
  JHU/APL, Bldg. 1
  Parsons Auditorium
  Mike Nemeth
  (301) 262-8140 (eves.)

- **MASSACHUSETTS**
  Boston FIG
  3rd Wed., 7 p.m.
  Bull HN
  300 Concord Rd., Billerica
  Gary Chanson ~~(617) 527-7206~~ (617) 899-4771

- **MICHIGAN**
  Detroit/Ann Arbor Area
  Bill Walters
  (313) 731-9660
  (313) 861-6465 (eves.)

- **MINNESOTA**
  MNFIG Chapter
  Minneapolis
  Fred Olson
  (612) 588-9532

- **MISSOURI**
  Kansas City Chapter
  4th Tues., 7 p.m.
  Midwest Research Institute
  MAG Conference Center
  Linus Orth (913) 236-9189

  St. Louis Chapter
  1st Tues., 7 p.m.
  Thornhill Branch Library
  Robert Washam
  91 Weis Drive
  Ellisville, MO 63011

- **NEW JERSEY**
  New Jersey Chapter
  Rutgers Univ., Piscataway
  Nicholas G. Lordi
  (908) 932-2662

- **NEW MEXICO**
  Albuquerque Chapter
  1st Thurs., 7:30 p.m.
  Physics & Astronomy Bldg.
  Univ. of New Mexico
  Jon Bryan (505) 298-3292

- **NEW YORK**
  Long Island Chapter
  3rd Thurs., 7:30 p.m.
  Brookhaven National Lab
  AGS dept.,
  bldg. 911, lab rm. A-202
  Irving Montanez
  (516) 282-2540

  Rochester Chapter
  Monroe Comm. College
  Bldg. 7, Rm. 102
  Frank Lanzafame
  (716) 482-3398

- **OHIO**
  Columbus FIG Chapter
  4th Tues.
  Kal-Kan Foods, Inc.
  5115 Fisher Road
  Terry Webb
  (614) 878-7241

  Dayton Chapter
  2nd Tues. & 4th Wed., 6:30 p.m.
  CFC
  11 W. Monument Ave. #612
  Gary Ganger (513) 849-1483

- **PENNSYLVANIA**
  Villanova Univ. Chapter
  1st Mon., 7:30 p.m.
  Villanova University
  Dennis Clark
  (215) 860-0700

- **TENNESSEE**
  East Tennessee Chapter
  Oak Ridge
  3rd Wed., 7 p.m.
  Sci. Appl. Int'l. Corp., 8th Fl.
  800 Oak Ridge Turnpike
  Richard Secrist (615) 483-7242

- **TEXAS**
  Austin Chapter
  Matt Lawrence
  PO Box 180409
  Austin, TX 78718

  Dallas Chapter
  4th Thurs., 7:30 p.m.
  Texas Instruments
  13500 N. Central Expwy.
  Semiconductor Cafeteria
  Conference Room A
  Warren Bean (214) 480-3115

Houston Chapter
3rd Mon., 7:30 p.m.
Houston Area League of PC Users (HAL-PC)
1200 Post Oak Rd.
(Galleria area)
Russell Harris
(713) 461-1618

- **VERMONT**
  Vermont Chapter
  Vergennes
  3rd Mon., 7:30 p.m.
  Vergennes Union High School
  RM 210, Monkton Rd.
  Hal Clark (802) 453-4442

- **VIRGINIA**
  First Forth of Hampton Roads
  William Edmonds
  (804) 898-4099

  Potomac FIG
  D.C. & Northern Virginia
  1st Tues.
  Lee Recreation Center
  5722 Lee Hwy., Arlington
  Joseph Brown
  (703) 471-4409
  E. Coast Forth Board
  (703) 442-8695

  Richmond Forth Group
  2nd Wed., 7 p.m.
  154 Business School
  Univ. of Richmond
  Donald A. Full
  (804) 739-3623

- **WISCONSIN**
  Lake Superior Chapter
  2nd Fri., 7:30 p.m.
  1219 N. 21st St., Superior
  Allen Anway (715) 394-4061

INTERNATIONAL

- **AUSTRALIA**
  Melbourne Chapter
  1st Fri., 8 p.m.
  Lance Collins
  65 Martin Road
  Glen Iris, Victoria 3146
  03/889-2600
  BBS: 61 3 809 1787

  Sydney Chapter
  2nd Fri., 7 p.m.
  John Goodsell Bldg., RM LG19
  Univ. of New South Wales
  Peter Tregeagle
  10 Binda Rd.
  Yowie Bay 2228
  02/524-7490
  Usenet:
  tedr@usage.csd.unsw.oz

- **BELGIUM**
  Belgium Chapter
  4th Wed., 8 p.m.
  Luk Van Loock
  Lariksdreef 20
  2120 Schoten
  03/658-6343

  Southern Belgium Chapter
  Jean-Marc Bertinchamps
  Rue N. Monnom, 2
  B-6290 Nalinnes
  071/213858

- **CANADA**
  Forth-BC
  1st Thurs., 7:30 p.m.
  BCIT, 3700 Willingdon Ave.
  BBY, Rm. 1A-324
  Jack W. Brown
  (604) 596-9764 or
  (604) 436-0443
  BCFB BBS (604) 434-5886

  Northern Alberta Chapter
  4th Thurs., 7–9:30 p.m.
  N. Alta. Inst. of Tech.
  Tony Van Muyden
  (403) 486-6666 (days)
  (403) 962-2203 (eves.)

  Southern Ontario Chapter
  Quarterly: 1st Sat. of Mar., June, and Dec. 2nd Sat. of Sept.
  Genl. Sci. Bldg., RM 212
  McMaster University
  Dr. N. Solntseff
  (416) 525-9140 x3443

- **ENGLAND**
  Forth Interest Group-UK
  London
  1st Thurs., 7 p.m.
  Polytechnic of South Bank
  RM 408
  Borough Rd.
  D.J. Neale
  58 Woodland Way
  Morden, Surry SM4 4DS

- **FINLAND**
  FinFIG
  Janne Kotiranta
  Arkkitehdinkatu 38 c 39
  33720 Tampere
  +358-31-184246

- **GERMANY**
  Germany FIG Chapter
  Heinz Schnitter
  Forth-Gesellschaft e.V.
  Postfach 1110
  D-8044 Unterschleißheim
  (49) (89) 317 37 84
  Munich Forth Box:
  (49) (89) 871 45 48
  8N1 300, 1200, 2400 baud
  e-mail uucp:
  secretary@forthev.UUCP
  Internet:
  secretary@Admin.FORTH-eV.de

- **JAPAN**
  Japan Chapter
  Toshio Inoue
  University of Tokyo
  Dept. of Mineral Development
  Faculty of Engineering
  7-3-1 Hongo, Bunkyo-ku
  Tokyo 113, Japan
  (81)3-3812-2111 ext. 7073

- **REPUBLIC OF CHINA**
  R.O.C. Chapter
  Ching-Tang Tseng
  P.O. Box 28
  Longtan, Taoyuan, Taiwan
  (03) 4798925

- **SWEDEN**
  SweFIG
  Per Alm
  46/8-929631

- **SWITZERLAND**
  Swiss Chapter
  Max Hugelshofer
  Industrieberatung
  Ziberstrasse 6
  8152 Opfikon
  01 810 9289

Forth is easy to implement, so many people do it. And they don't just type in fig-Forth listings, they experiment.

See "Best of GEnie"

- **HOLLAND**
  Holland Chapter
  Maurits Wijzenbeek
  Nieuwendammerdijk 254
  1025 LX Amsterdam
  The Netherlands
  ++(20) 636 2343

- **ITALY**
  FIG Italia
  Marco Tausel
  Via Gerolamo Forni 48
  20161 Milano

SPECIAL GROUPS

- **Forth Engines Users Group**
  John Carpenter
  1698 Villa St.
  Mountain View, CA 94041
  (415) 960-1256 (eves.)

(Fast Forthward, continued from page 31.)

ceeding full speed ahead to create such boundary-breaking standards (see "Apple Event Objects and You" in the May issue of *develop*, Apple's magazine for developers).

Sometimes complexity is not the obstacle to progress. In cases like these, the added complexity of implementation may be of little or no detriment to progress. Rather than ugly complexity, I see a rich tapestry shaping up that will confer significant value to many new and existing computer users. These layers of software can work like agents, interceding on the behalf of users so that they do not have to deal directly with tedious supporting technologies such as local networks and wide-area electronic mail services.

Correct me if I am wrong, but I think these services will make computers sell as well as telephones. If the market for hardware is already saturated, there will still be vast amounts of software extensions to be sold.

For Forth to be able to get a piece of that market, however, we cannot continue to view ourselves as iconoclasts in perpetual defiance of companies large enough to drive such standardization efforts. Even our attitudes toward our own standards, such as Forth-83 and ANS Forth, are too much of the "necessary evil" sort. Let's make the effort to see the value of Forth language standards, GUI standards, and operating-system-extensions standards. (Isn't it odd how "operating-system-extensions standards" almost sounds like it came out of a Forth textbook?)

ANS Forth will certainly be one step in the right direction. Forth's ability to support libraries, as well as its ability to take advantage of C libraries, would be helpful too.

We must realize that the needs of the user dictate what software will sell. The GUI makes the computer less taxing and more enjoyable to use. If it makes life more difficult for the programmer, the consumer couldn't care less. (Computer users regularly buy software without giving any thought to the underlying language used to create it.)

If we fail to grasp that user needs drive sales of products, we forfeit our chances of any real commercial success. I am starting to think that some of us have mistaken our own desires, as developers, for user needs. (Even if Apple and IBM do forfeit the future, how likely is it that ordinary computer users will dabble with Forth and discover that it meets their computing needs better?)

From a marketing perspective, Forth is best able to compete when short-run or one-of-a-kind products must be created. Big companies are not interested in the returns possible from such undertakings. Nevertheless, small but highly innovative engineering outfits often take great pride in designing such short-run, custom products.

But does Forth's ability to compete in small markets translate into an ability to compete in larger markets? Please show me that it can—but please do so on the basis of the customer needs in those larger markets.

Your thoughts about Forth's markets are valued. Can one of you be persuaded to write a regular "Forth Developer Opportunities" column? This magazine is yours to create. I eagerly await your Fast Forthward essay—whether its focus is marketing or a technical subject. (For any Forth vendors amongst you, this request applies doubly.)

—Mike Elola

(GEnie, continued from page 36.)

> Addressable stacks are processor-dependent (i.e., *not* present in all Forth systems and *not* portable).
>
> Implementation-dependent means "not present in all Forth systems." Processor-dependent means not implementable on all processors (e.g., memory > 64K).

(e.g., a Forth processor in silicon that keeps TOS, etc. in internal registers that have no data bus addresses. Implementation- vs. processor-dependent are fuzzy distinctions to make with Forth silicon on the scene.)

> The structure of the Forth dictionary is implementation-dependent. The implementation will depend on the underlying hardware, therefore the structure of the dictionary is indirectly processor-dependent.
>
> You are saying that there are processors that cannot process linked lists (or hash tables or an ADT for the dictionary). Don't make me laugh so hard!

Laugh? Go ahead if you need to. There is quite a bit more structure to a dictionary than a linked list. I haven't seen the results of the internals ADT for the dictionary, but if it is really an ADT, it doesn't prescribe the structure of the dictionary, merely its interface. The structure of the dictionary, on segmented architectures, may be segmented. On systems tight on RAM, the structure might even allow swapping, so that not all parts of the dictionary entries are even in memory at the same time. Perhaps you have a different idea of what structure is?

> I think it would help to be more precise here. There is a difference between *not* saying that something *is* portable, and making something non-portable.
>
> There may be a difference, but it does not matter. If nothing else happens, anything not covered by the standard will be non-portable.

Anything not covered by the standard is not going to be any more non-portable than it was before. The standard doesn't insist that every aspect not covered by it be implemented incompatibly.

To participate in the up-to-the-minute discussion between on-line Forth users, and to gain access to Forth software libraries, see the GEnie logon procedure on page 39.

The Fireman Syndrome

Conducted by Russell L. Harris
Houston, Texas

Returning once more to the vein of the two previous columns, i.e., to the matter of finding commercial opportunities for programming in Forth, I offer for your consideration the following thesis: Perhaps we spend too much time trying to find ways to make ourselves competitive, when, possessing via Forth a rather unique capability, we really should be seeking a niche in which we have no competition. In like manner, instead of attempting to demonstrate the superiority of Forth, it may prove a wiser course of action to search out areas in which Forth is simply indispensable—areas in which program complexity makes impractical the use of assembler, and in which other considerations preclude the use of C, Ada, and other high-level languages.

One Man's Trash, Another Man's Treasure

I remember reading of the outfit which bought up germanium transistor fabrication lines, one by one, as the various semiconductor foundries discontinued production of germanium devices. The company was, with a single exception, able to purchase at scrap prices every germanium line in the world; they were outbid on only one sale. Having acquired a virtual monopoly, they now have all the business they can handle, for it seems that the demand for germanium transistors, although not growing, is nevertheless quite stable. The stability of the market is the consequence of three factors, the first of which is that germanium devices have an operating life much shorter than that of silicon devices (Surprise! Transistors do *not* have an infinite service life!) and thus require replacement every few years. The second factor is that a surprisingly large number of circuits in current use were designed around germanium devices, and it is simply not economical to redesign them, so long as replacement transistors can be had. Finally, some circuits depend upon the electrical characteristics of germanium transistors to the extent that there is no practical silicon alternative. What a niche!

Pride & Prejudice

In the movie January Man, the hero describes his job as a fireman: "Burning building... everyone else runs out... you run in. Basically, it's a maniac's job." The fireman is a valued and respected member of society, for he has a specialized and vital function at which he is competent. Yet, the nature of the work attracts but few.

There is an analogous role in the programming world, one which Forth programmers generally are well qualified to fill. By aiming at work which is outside the capability or inclination of other programmers, it is possible both to program in Forth and to earn a respectable living in the process. In the words of the old country song, it is a way to "have your Kate, and Edith, too."

The next time you are with a client, ask where his biggest headaches lie, and what tasks his people find most onerous. Chances are, you will find that the mindless, headlong stampede into the abyss of C has left his shop devoid of programmers proficient in the areas of real-time programming and ROM-based systems. Even those knowledgeable in assembly coding of embedded systems may be loath to accept such projects, fearful that a display of competence may relegate them forevermore to mundane duties, with no hope of return to the esoteric realm of C. Every such shop is a potential client for a Forth programmer.

If at First You Don't Succeed...

Inasmuch as I have yet to detect reader feedback from my first column, I must assume a dearth of interest regarding demonstration apparatus. This being the case, a shift in emphasis is in order. Accordingly, I plan to devote the next and subsequent columns to matters relating to the programming of embedded systems.

In the course of the upcoming odyssey, we shall take a tutorial approach to a number of pertinent topics, delving into the morass of metacompilation and attempting to divest it of its enigmatic repute. Within the realm of subjects to be considered are cross-compilation, both batch and interactive, and the creation of new Forth systems via assembler code.

Reader feedback, regarding areas of particular interest and areas in which confusion abounds, will aid in charting the course of study.

R.S.V.P.

Russell Harris is a consulting engineer with experience in a variety of fields. He particularly enjoys 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.

The language market no longer promises windfall profits. The good side of this is that software source and idea piracy is no longer a major risk. Perhaps now a new vendor strategy is possible...

See "Letters" in this issue.

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 October 1, 1992 to: FORML Conference, Forth Interest Group, P.O. Box 2154, Oakland, CA 94621

Completed papers are due November 1, 1992.

Conference Registration

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

Conference attendee in double room—$365 • Non-conference guest in same room—$225 • Children under 18 years old in same room—$155 • Infants under 2 years old in same room—free • Conference attendee in single room—$465

Complete registration by October 15, 1992 and Forth Interest Group members and their guests are eligible for a ten percent discount on registration fees.

Register by calling the Forth Interest Group business office at (510) 893-6784 or writing to: FORML Conference, Forth Interest Group, P.O. Box 2154, Oakland, CA 94621

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

Second Class
Postage Paid at
San Jose, CA