$10 Volume XX, Number 2 July 1998 August

FORTH DIMENSIONS

Forth to the IRS

EuroForth 98 Conference

Two Problems in ANS Forth

Safer Numeric Input

Finite State Machines

Greetings from the FIG office!

This is the year the FORML Conference turns 20! What a milestone. We already have participants giving us titles for the talks they are planning to present. If you'd like to see what we've got lined up, just log on to the Forth Interest Group web site (www.forth.org), scroll to Forth Conferences, and there you have it. By now, you should have received a purple flyer with the FORML registration information. If not, you can check our web site for that information, too, or contact the office and we will be happy to mail, fax, or e-mail the registration information to you. In addition to the regular registration, there are three levels of increased financial sponsorship for FORML available: Bronze, Silver, and Gold. If you or your company would like to become a FORML sponsor, and/or have equipment and/or services you would like to offer, please contact the office and we will give you the details. We are looking forward to seeing you in November!

The Forth Interest Group is kept running with your membership dues and kind donations. We continue to add new members every month; however, we also lose members. We need your help to steadily increase our membership. If you do have friends or colleagues who are interested in joining, please contact us at the office and we will be happy to send membership information along with a complimentary issue of *Forth Dimensions* for their review.

Just a reminder, it is important to renew early. We don't want you to miss an issue of *Forth Dimensions*. If you have any questions as to when your membership expires, just contact us—Eddy or I will be happy to help you with that information. And it is always printed above your name on the address label that comes with your *Forth Dimensions*. When you do renew late, we try to make sure that you don't miss issues, but it is an added expense and is labor-intensive to do that. While you're renewing your membership, please consider increasing your level of membership and/or making an additional contribution to your organization. Every bit helps, and so many of you are very generous in this way. It is greatly appreciated.

Also take a moment now to look at the mail-order form in the center of this issue. Isn't there something you have been thinking about getting? There is a wide selection of books and software for you to purchase. Some of these items are in very limited supply, and we encourage you to purchase them before the Christmas rush. And remember, members get an additional 10% off their purchases!

We are continuing to update our membership database. If you have not received an e-mail from the FIG office, there is a good chance that we do not have your e-mail address in our database. Please take a moment and send us your e-mail address. Remember to keep us informed when your address and/or phone number changes. Speaking of which, like many of you, our telephone area code has changed. Our old area code was 408, and our new area code is 831. Please make a note of it in your files.

And remember—together we can make a difference!

Cheers,

Trace Carter, Administrative Manager
Forth Interest Group
100 Dolores Street, Suite 183
Carmel, CA 93923 USA
voice: 831.373.6784 • fax: 831.373.2845
e-mail: office@forth.org

CONTENTS

DEPARTMENTS

EDITORIAL

ANS Forth Revisited

As promised in our last issue, this time we introduce refereed material from the *Journal of Forth Applications and Research*. Since *JFAR* has gone electronic, we anticipate that *FD* will be the primary printed source of its refereed Forth articles. It is with special pleasure that we inaugurate this series with Julian V. Noble's "Finite State Machines in Forth," and we look forward to presenting equally informative and challenging material in the future.

ANS Forth, and its ISO/IEC equivalent, has enjoyed a surprisingly smooth road to general acceptance. Those who remember some of the past tumult over standards had predicted more difficulty. This is not to say that it has been adopted universally. The standard has not been entirely without detractors, and even its proponents will mention things which would be useful in the standard but which could not be agreed upon at the time it was passed.

But I believe most people recognize that the standard is both a good way for us to communicate among ourselves, to formalize standard practice, and to achieve some greater degree of legitimacy in the world of business. The ANS effort was first discussed at a FORML Conference and, at this year's conference, the standard will be re-opened for evaluation and possible revision. Participants are welcome, and *FD* naturally will report the details.

Meanwhile, as we continue striving to accelerate our publishing schedule in order to get our issues out on time, please enjoy this issue's content—and let the included writers guidelines inspire you to help us by sharing your own discoveries, opinions, and techniques with your fellow readers. I'll look forward to hearing from you.

—Marlin Ouverson, Editor

...Forth programs should have a tab indent of 40 spaces. That way we won't be tempted to write huge multi-level control structures, and will want to factor a little.

—Andrew McKewan (from comp.lang.forth)

Forth Dimensions
Volume XX, Number 2
July 1998 August

Published by the
Forth Interest Group

Editor
Marlin Ouverson

Circulation/Order Desk
Trace Carter

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

Subscription to *Forth Dimensions* is included with membership in the Forth Interest Group at $45 per year (U.S.) $60 (international). For membership, change of address, and to submit items for publication, the address is:

Forth Interest Group
100 Dolores Street, suite 183
Carmel, California 93923
Administrative offices:
408-37-FORTH Fax: 408-373-2845

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. FIG provides a climate of intellectual exchange and benefits intended to assist each of its members. Publications, conferences, seminars, telecommunications, and area chapter meetings are among its activities.

FORTH DIMENSIONS (ISSN 0884-0822) is published bimonthly for $45/60 per year by Forth Interest Group at 1340 Munras Avenue, Suite 314, Monterey CA 93940. Periodicals postage rates paid at Monterey CA and at additional mailing offices.

POSTMASTER: Send address changes to FORTH DIMENSIONS, 100 Dolores Street, Suite 183, Carmel CA 93923-8665.

Finite State Machines in Forth

Editor's note: an appendix addressing compatibility with ANS Forth is scheduled for our next issue and to be posted with the on-line version of this paper at http://dec.bournemouth.ac.uk/forth/jfar/vol7/paper1/paper.html

This note provides methods for constructing deterministic and non-deterministic finite state automata in Forth. The "best" method produces a one-to-one relation between the definition and the state table of the automaton. An important feature of the technique is the absence of (slow) nested IF clauses.

1. Introduction

Certain programming problems are difficult to solve procedurally even using structured code, but simple to solve using abstract finite state machines (FSMs) [1]. For example, a compiler must distinguish a text string representing, say, a floating-point number, from an algebraic expression that might well contain similar characters in similar order. Or a machine controller must select responses to predetermined inputs that occur in random order.

Such problems are interesting because a program that responds to indefinite input is closer to a "thinking machine" than a mere sequential program. Thus, a string that represents a floating-point number is defined by a set of rules; it has no definite length, nor do the symbols appear in a definite order. Worse, more than one form for the same number may be permissible—user-friendliness demands a certain flexibility of format.

Although generic pattern recognition can be implemented through logical expressions (i.e., by concatenating sufficiently many Ifs, ELSEs, and THENs) the resulting code is generally hard to read, debug, or modify. Worse, this approach is anything but structured, no matter how "prettily" the code is laid out: indentation can only do so much. And programs consisting mainly of logical expressions can be slow, because many processors dump their pipelines upon branching [2]. These defects of the nested-IF approach are attested to by the profusion of commercial tools to overcome them: Stirling Castle's Logic Gem (that translates and simplifies logical expressions), Matrix Software's Matrix Layout (that translates a tabular representation of an FSM into one of several languages such as BASIC, Modula-2, Pascal, or C), or AYECO, Inc.'s COMPEDITOR (that performs a similar translation). *[These CASE tools were available at least as recently as 1993 from The Programmer's Shop and other developer-oriented software discounters.]*

Forth is a particularly well-structured language that encourages natural, readable ways to generate FSMs. This note describes several high-level Forth implementations. Finite state machines have been discussed previously in this journal [3, 4]. The present approach improves on prior methods.

2. A Simple Example

Consider the task of accepting numerical input from the keyboard. An unfriendly program lets the user enter the entire number before informing him that he typed two decimal points after the first digit. A friendly program, by contrast, refuses to recognize or display illegal characters. It waits instead for a legal character or carriage return (signifying the end of input). It permits backtracking, allowing erasure of incorrect input.

To keep the example small, our number-input routine allows signed decimal numbers without power-of-10 exponents (fixed point, in FORTRAN parlance). Decimal points, numerals, and leading minus signs are legal, but no other ASCII characters (including spaces) will be recognized. Here are some examples of legal numbers:

0.123, .123, 1.23, -1.23, 123, etc.

From these examples we derive the rules:

- Characters other than 0–9, -, and . are illegal.
- Numerals 0–9 are legal.
- The first character can be -, 0–9, or a decimal point.
- After the first character, - is illegal.
- After the first decimal point, decimal points are illegal.

A traditional procedural approach might look something like Listing One.

Listing One

```
VARIABLE PREVIOUS.MINUS?      \ history semaphores
VARIABLE PREVIOUS.DP?

: DIGIT?    ( c -- f)   ASCII 0  ASCII 9 WITHIN  ;      \ tests
: DP?       ( c -- f)   ASCII .  =  ;
: MINUS?    ( c -- f)   ASCII -  =  ;
: FIRST.MINUS?   MINUS?  PREVIOUS.MINUS?  @   NOT  AND   ;
: FIRST.DP?      DP?     PREVIOUS.DP?     @   NOT  AND   ;
: LEGAL?    ( c -- f)         \ horrible example
              DUP  DIGIT?
              IF    DROP  TRUE  DUP   PREVIOUS.MINUS?  !
              ELSE  DUP    FIRST.MINUS?
                    IF     DROP   TRUE  DUP   PREVIOUS.MINUS?  !
                    ELSE   FIRST.DP?
                           IF    TRUE  DUP   PREVIOUS.DP?  !
                           ELSE  FALSE
                           THEN
                    THEN
              THEN  ;
```

J.V. Noble • Charlottesville, Virginia
jvn@fermi.clas.virginia.edu

Listing Two

```
: Getafix
    FALSE  PREVIOUS.MINUS?  !    FALSE  PREVIOUS.DP?  !
                \ initialize history semaphores
    BEGIN  KEY  DUP  CR <>   WHILE
        LEGAL?   IF   DUP  ECHO  APPEND   THEN
        REPEAT  ;
```

The word that does the work (with apologies to Uderzo and Goscinny, creators of Asterix) is shown in Listing Two.

What makes this example—whose analogs appear frequently in published code in virtually every language—horrible? Each character whose legality is time-dependent requires a history semaphore. It is therefore difficult to tell by inspection that the word LEGAL?'s logic is actually incorrect, despite the simplification obtained by partial factoring and logical arithmetic.

3. Forth Finite State Machines

The FSM approach replaces the true/false historical semaphores with one state variable. The rules can be embodied in a state table that expresses the response to each possible input in terms of a concrete action and a state transition, as shown in Figure One.

In the state table,

- The illegality of "other" characters is expressed by the uniform action X and the absence of state transitions.
- The special status of the first character is expressed by the fact that all acceptable characters lead to transitions out of the initial state (0).
- An initial - sign or digit leads to state 1, where a - sign is unacceptable.
- A decimal point always moves the system to state 2, where decimal points are not accepted.

While some FSMs can be synthesized with BEGIN ... WHILE ... REPEAT or BEGIN ... UNTIL loops, keyboard input does not readily lend itself to this approach. We now explore three implementations of the state table of Figure One as Forth FSMs.

3.1. Brute-Force FSM

The "brute-force" FSM uses the Eaker CASE statement, either in its original form [5] or with a simplified construct from HS/Forth [6]. HS/Forth provides defining words CASE: ... ;CASE whose daughter words execute one of several words in their definition, as in:

```
CASE:
  CHOICE   WORD0  WORD1  WORD2  WORD3 ... WORDn
;CASE
3 CHOICE    ( executes WORD3 )  ok
```

HS/Forth's CASE: ... ;CASE incurs virtually no run-time speed penalty relative to executing the words themselves. Now, how do we use CASE: ... ;CASE to implement an FSM? First we need a state variable (initialized to 0) that can assume the values 0, 1, and 2. To test whether an input character is a numeral, minus sign, decimal point, or "other," we define the words in Listing Three *[Note: the ANSI Standard [7] renames ASCII to CHAR and UNDER to TUCK; also, DDUP is specific to HS/Forth and should be replaced with 2DUP for ANSI compliance. WITHIN as used here returns TRUE if* $a \le n \le b$, *which is different from the ANS specification. These remarks apply here and below, except as noted.]:*

Now, to use CASE: ;CASE we define three words to handle the tests in each state; see Listing Four.

Finally, in Listing Five we define the words that use the definitions given in Listing Four.

3.2. A Better FSM

While the approach outlined above in §3.1 (essentially the method described recently by Berrian [8]) both works and produces much clearer code than the binary logic tree of §2, it nevertheless can be improved. The words (0), (1), and (2) are inadequately factored (they contain the tests performed on the input character). They also contain IF ... ELSE ... THEN branches (which we prefer to avoid for the sake of speed and structure). Finally, each FSM must be hand crafted from numerous subsidiary definitions.

We want to translate the state table in Figure One into a program. The preceding attempt was too indirect—each state was represented by its own word that did too much. Perhaps we can achieve the desired simplicity by translating more directly. In Forth, such translations are most naturally accomplished via defining words. Suppose we visualize the state table as a matrix, whose cells contain action specifications (addresses or execution tokens), whose columns represent input categories, and whose rows are states. If we translate input categories to column numbers, the category and the current value of the state variable (row index) determine a unique cell address, whose content can be fetched and executed.

Translating the input to a column number factors the tests into a single word that executes once per character. This word should avoid time-wasting branching instructions, so all decisions (as to which cell of the table to EXECUTE) will be computed rather than decided. For our test example, the preliminary definitions are given in Listing Six and the input translation is carried out by the definitions in Listing Seven.

Now we must plan the state-table compiler. In general, we define an action word for each cell of the table that will perform the required action and state change. At compile time,

Figure One

Input	OTHER?			DIGIT?			MINUS?			DP?		
State	Does		Trans	Does		Trans	Does		Trans	Does		Trans
0	X	→	0	E	→	1	E	→	1	E	→	2
1	X	→	1	E	→	1	X	→	1	E	→	2
2	X	→	2	E	→	2	X	→	2	X	→	2

Figure One. State table summarizing the rules for fixed-point numbers. E stands for *echo* (to the CRT) and X for *do nothing*.

Listing Three

```
VARIABLE mystate    mystate 0!
: WITHIN   ( n a b -- f) DDUP MIN  -ROT  MAX  ROT
           UNDER MIN  -ROT MAX  = ;
: DIGIT?  ( c -- f )   ASCII 0  ASCII 9 WITHIN  ;
: DP?     ( c -- f )   ASCII .  = ;
: MINUS?  ( c -- f )   ASCII -  = ;
```

Listing Four

```
: (0)     ( char -- )   DUP
          DIGIT?   OVER   MINUS?   OR
          IF   EMIT   1 mystate !   ELSE   DUP   DP?
          IF   EMIT   2 mystate !   ELSE   DROP   THEN    THEN  ;

: (1)     ( char -- )   DUP   DIGIT?
          IF   EMIT   1 mystate !   ELSE   DUP  MINUS?
          IF   1 mystate !          ELSE   DUP   DP?
          IF   EMIT   2 mystate !   ELSE   DROP
          THEN    THEN     THEN  ;

: (2)     ( char -- )   DUP
          DIGIT?   IF   EMIT   ELSE   DROP   THEN   ;
```

Listing Five

```
CASE:   <Fixed.Pt#>    (0)  (1)  (2)   ;CASE

: Getafix     0 mystate !                    \ initialize state
              BEGIN
                  KEY   DUP   13 <>          \ not CR ?
              WHILE   mystate @   <Fixed.Pt#> \ execute FSM
              REPEAT  ;
```

Listing Six

```
VARIABLE mystate   0 mystate !

: WITHIN   ( n a b -- f) DDUP MIN  -ROT  MAX
           ROT TUCK MIN  -ROT MAX  = ;

: DIGIT?  ( n -- f )   ASCII 0  ASCII 9 WITHIN  ;
: DP?  ASCII .  = ;
: MINUS?  ASCII -  = ;
```

Listing Seven

```
: cat->col#   ( n -- n')
         DUP   DIGIT?   1 AND                  \ digit   -> 1
         OVER  MINUS?   2 AND  +               \ -       -> 2
         SWAP  DP?      3 AND  +               \ dp      -> 3
;                                              \ other   -> 0
```

Listing Eight

```
: TUCK       COMPILE UNDER  ;                  \ ANS compatibility
: WIDE   ;                                     \ NOOP for clarity
: CELLS      COMPILE  2*  ;                    \ ANS compatibility
: CELL+      COMPILE  2+  ;                    \ ANS compatibility
: PERFORM    COMPILE @   COMPILE EXECUTE  ;    \ alias
: FSM:       ( width -- )  CREATE  ,  ]
       DOES>  ( n adr -- )
              TUCK  @  mystate @  *   +   CELLS   CELL+  +
              ( adr')  PERFORM  ;
```

the defining word will compile an array of the execution addresses (execution tokens in ANS-Forth parlance [9]) of these action words. At run time, the child word computes the address of the appropriate matrix cell from the user-supplied column number and the current value of `mystate`, fetches the execution address from its matrix cell, and EXECUTEs the appropriate action. Since a table can have arbitrarily many columns, the number of columns must be supplied at compile time. These requirements lead to the definitions in Listing Eight.

Here CREATE makes a new header in the dictionary, , stores the top number on the stack in the first cell of the parameter field, and] switches to compile mode. The run-time code computes the address of the cell containing the vector to the desired action, fetches that vector, and executes the action. *[Note: This simple and elegant implementation only works with indirect-threaded Forths. An ANSI Standard alternative is provided in the Appendix.]*

Now we apply this powerful new word to our example problem. From Figure One we see that transitions (changes of state) occur only in cells (0,0), (0,1), (0,2), (1,0), and (1,2). These are always associated with EMIT (E in the figure). No change of state accompanies a wrong input, and the associated action is to DROP the character. There are, thus, only two distinct state-changing actions we need define:

```
: (00)   EMIT  1 mystate ! ;
: (02)   EMIT  2 mystate ! ;
```

Since we must test for four conditions on the input character, the state table will be four columns wide:

```
4 WIDE FSM: <Fixed.Pt#>    ( action# -- )
\          other    num     -      .      \ state
           DROP     (00)   (00)   (02)    \ 0
           DROP     (00)   DROP   (02)    \ 1
           DROP     (02)   DROP   DROP  ; \ 2
```

The word that does the work is:

```
: Getafix
  0 mystate !
  BEGIN   KEY   DUP   13 <>   \ not CR
  WHILE   DUP   cat->col#   <Fixed.Pt#> REPEAT
;
```

We can immediately test the FSM, as follows:

```
FLOAD F:X.1  Loading F:X.1  ok
ASCII 3 cat->col# . 1  ok
ASCII 0 cat->col# . 1  ok
ASCII - cat->col# . 2  ok
ASCII . cat->col# . 3  ok
ASCII A cat->col# . 0  ok

: Getafix
  0 mystate !
    BEGIN  KEY  DUP  13 <> WHILE
    DUP   cat->col#   <Fixed.Pt#> REPEAT ;  ok
Getafix -3.1415975 ok
Getafix 55.3259 ok
```

The incorrect input of excess decimal points, incorrect minus signs, or non-numeric characters does not show because, as intended, they were dropped without echoing to the screen.

3.3. An Elegant FSM

The defining word FSM: of §3.2, while useful, nevertheless has room for improvement. This version hides the state transitions within the action words compiled into the child word's cells. A more thoroughly factored approach would explicitly specify transitions next to the actions they follow, within the definitions of each FSM. Definitions will become more readable, since each looks just like its state table; that is, our ideal FSM definition will look like Listing Nine, so we would never need the words (00) and (02). Fortunately, it is not hard to redefine FSM: to include the transitions explicitly. We may use CONSTANTs to effect the state transitions:

```
0 CONSTANT >0
1 CONSTANT >1
2 CONSTANT >2
...
```

and modify the run time portion of FSM: accordingly (Listing Ten).

Note that we have defined the run-time code so that the change in state variable precedes the run-time action. Sometimes the desired action is an ABORT and an error message. Changing the state variable first lets us avoid having to write a separate error handler for each cell of the FSM, yet we can tell where the ABORT took place. If the ABORT were first, the state would not have been updated.

The FSM is then defined as in Listing Eleven, which is clear and readable. Of course, one could define the run-time (DOES>) portion of FSM: to avoid the need for extra CONSTANTs >0, >1, etc. The actual numbers could be stored in the table via the code in Listing Twelve.

However, for reasons given in §4 below, it is better to implement the state transition with CONSTANTs rather than with numeric literals.

4. The Best FSM So Far

Experience using the FSM approach to write programs has motivated two further improvements. First, suppose one needs to nest FSMs, i.e., to compile one into another, or even to RECURSE. The global variable mystate precludes such finesse. It therfore makes sense to include the state variable for each FSM in its data structure, just as with its WIDTH. This modification protects the state of an FSM from any accidental interactions, at the cost of one more memory cell per FSM, since if the state has no name it cannot be invoked. Second, suppose one or other of the action words is supposed to leave something on the stack, and that, for some reason, it is desirable to alter the state after the action rather than before (this is, in fact, the more natural order of doing things). Since there is no way to know in advance what the stack effect will be, we use the return stack for temporary storage, to avoid collisions. The revision is, thus, as shown in Listing Thirteen.

The revised keyboard input word of our example is

```
: Getafix
   0  ' <Fixed.Pt#> !
     BEGIN   KEY   DUP   13 <>       WHILE
     DUP   cat->col#  <Fixed.pt#>   REPEAT ;
```

Note that the state variable is initialized to zero because we know it is stored in the first cell in the parameter field of the FSM which can be accessed by the phrase ' <Fixed.Pt#> (or the equivalent in the Forth dialect being used—see Appendix).

5. Non-deterministic Finite State Machines

We are now in a position to explain why defining a CONSTANT to manage the state transitions is better than merely incorporating the next state's number. First, there is no obvious reason why states cannot be named rather than numbered. The Forth outer interpreter itself is a state machine with two states, COMPILE and INTERPRET; i.e., names are often clearer than numbers.

The use of a word rather than a number to effect the transition permits a more far-reaching modification. Our code defines a compiler for deterministic FSMs, in which each cell in the table contains a transition to a definite next state. What if we allowed branching to any of several next states, following a given action? FSMs that can do this are called non-deterministic. Despite the nomenclature, and despite the misleading descriptions of such FSMs occasionally found in the literature [10], there need be nothing random or "guess-like" about the next transition. What permits multiple possibilities is additional information, external to the current state and current input.

Here is a simple non-deterministic FSM used in my FORmula TRANslator [11]. The problem is to determine whether a piece of text is a proper identifier (that is, the name of a variable, subroutine, or function) according to the rules

Listing Nine

```
4 WIDE FSM: <Fixed.Pt#>
\ input:  |  other?  |  num?   |  minus?  |   dp?     |
\ state:  ----------------------------------------------
   ( 0 )     DROP  0    EMIT  1   EMIT  1     EMIT  2
   ( 1 )     DROP  1    EMIT  1   DROP  1     EMIT  2
   ( 2 )     DROP  2    EMIT  2   DROP  2     DROP  2 ;
```

Listing Ten

```
: FSM:   ( width -- )   CREATE  ,  ]   DOES>   ( col# -- )
    TUCK   @                            ( -- adr col# width )
    mystate @  *  +  2*  CELLS  CELL+  +  ( -- offset )
    DUP   CELL+  PERFORM   mystate !   PERFORM   ;
```

Listing Eleven

```
4 WIDE FSM: <Fixed.Pt#>
\ input:  |  other?  |  num?   |  minus?  |   dp?     |
\ state:  ----------------------------------------------
   ( 0 )     DROP >0    EMIT >1   EMIT >1     EMIT >2
   ( 1 )     DROP >1    EMIT >1   DROP >1     EMIT >2
   ( 2 )     DROP >2    EMIT >2   DROP >2     DROP >2 ;
```

Listing Twelve

```
4 WIDE FSM: <Fixed.Pt#>
\ input:  |   other?    |   num?    |   minus?   |    dp?     |
\ state:  ------------------------------------------------------
   ( 0 )   DROP [  0 , ]  EMIT [  1 , ] EMIT [  1 , ] EMIT [  2 , ]
   ( 1 )   DROP [  1 , ]  EMIT [  1 , ] DROP [  1 , ] EMIT [  2 , ]
   ( 2 )   DROP [  2 , ]  EMIT [  2 , ] DROP [  2 , ] DROP [  2 , ] ;
```

Listing Thirteen

```
: 2@     COMPILE  D@  ;           \ alias
: WIDE   0 ;
: FSM:   ( width 0 -- )
      CREATE   ,  ,   ]
      DOES>                ( col# adr -- )
         DUP >R  2@  *   + ( -- col#+width*state )
         2*  2+  CELLS     ( -- offset-to-action)
         DUP >R            ( -- offset-to-action)
         PERFORM           ( ? )
         R> CELL+          ( -- offset-to-update)
         PERFORM           ( -- state')
         R> !   ;          \ update state
```

of FORTRAN. An ID must begin with a letter, and can be up to seven characters long, with characters that are letters or digits. To accelerate the process of determining whether an ASCII character code represents a letter, digit, or "other," we define a decoder (fast table translator):

```
: TAB:    ( #bytes -- )
          CREATE   HERE  OVER  ALLOT
          SWAP  0 FILL  DOES>  +  C@  ;
```

and a method to fill it quickly:

```
: install       ( col# adr char.n char.1 -- )
\ fast fill
                SWAP 1+ SWAP
                DO  DDUP I +  C!  LOOP  DDROP ;
```

The translation table we need for detecting IDs is

```
128 TAB: [ id]
1 ' [ id]   ASCII Z  ASCII A  install
1 ' [ id]   ASCII z  ASCII a  install
2 ' [ id]   ASCII 9  ASCII 0  install
```

This follows the Forth-79 convention that ' returns the PFA. To convert to Forth-83 or ANS, replace ' by ' >BODY .

Thus, e.g.:

```
ASCII R  [ id]  .  1 ok
ASCII s  [ id]  .  1 ok
ASCII 3  [ id]  .  2 ok
ASCII +  [ id]  .  0 ok
\ to convert to ANSI replace ASCII by CHAR
```

Now how do we embody the ID rules in an FSM? Our first attempt might look like Listing Fourteen.

To make sure the *id* is at most seven characters long, we had to provide eight states. The table is rather repetitious. A simpler alternative uses a VARIABLE to count the characters as they come in. The revision is shown in Listing Fifteen.

The resulting FSM is non-deterministic because the word >1? induces a transition either to state 1 or to (the terminal) state 2. It has fewer states and consumes less memory, despite the extra definitions. Because we have stuck to subroutines for mediating state transitions, going from a definite transition (via a CONSTANT such as >0 or >1) to an indefinite one requires no redefinitions.

The FSM in Figure Sixteen detects properly formed floating-point numbers. The FSM (fp#) does not count digits in the mantissa, but limits those in the exponent to two or fewer. A non-deterministic version of (fp#) reduces the number of states, while counting digits in both the mantissa and exponent of the number (Listing Seventeen).

The task of fleshing out the details—specifically, the words +mant, +exp, ?+1, >0?, >1?, and >3?—is left as an exercise for the reader.

6. Acknowledgments

I am grateful to Rick Van Norman and Lloyd Prentice for positive feedback about applications of the FSM compiler in areas as diverse as gas pipeline control and educational computer games.

7. Appendix

Here is a high-level definition of the HS/Forth CASE: ...;CASE (albeit it imposes more overhead than HS/Forth's version) that works in indirect-threaded systems:

```
: CASE:
     CREATE  ]
     DOES>   ( n -- )  OVER  + +  @  EXECUTE ;

: ;CASE   [ COMPILE] ;
     ;  IMMEDIATE    ( or just use ; )
```

However, it will not work with a direct-threaded Forth like F-PC. It fails because what is normally compiled into the body of the definition (by using] to turn on the compiler) in a direct-threaded system is not a list of execution tokens. The simplest alternative (it also works with indirect-threaded systems) factors the compilation function out of CASE:

```
: CASE:       CREATE   ;
: |      '  ,  ;           \ F83 and ANS version
: ;CASE                   \ no error checking
      DOES>  OVER + +  PERFORM  ;
```

Here is a usage example:

```
CASE: TEST    | *   | /   | +   | -   ;CASE
 3 4 0 TEST . <u>12 ok</u>
12 4 1 TEST . <u> 3 ok</u>
 5 7 2 TEST . <u>12 ok</u>
 5 7 3 TEST . <u>-2 ok</u>
```

Although the CASE statement can be made to extend over several lines if one likes, readability and good factoring suggest such definitions be kept short.

The same technique can be used to provide a version of FSM: that works with F-PC and other F83-based or ANS-compliant systems, for those who want to experiment with FSM: but lack HS/Forth to try the version §4 with. The definitions are:

```
: |      '  ,  ;        \ F83 and ANS version
: WIDE   0 ;

: FSM:    ( width 0 -- )       CREATE   ,  ,   ;
: ;FSM    DOES>                  ( col# adr -- )

  DUP >R   2@  *  +        ( -- col#+width*state )

           2*  2+  CELLS  ( -- offset-to-action)

           DUP >R        ( -- offset-to-action)
           PERFORM       ( ? )
           R> CELL+      ( -- ? offset-to-update)
           PERFORM       ( -- ? state')

           R> !   ;   ( ? )        \ update state
```

The FSM of §3.3 now takes the form of Listing Eighteen.

8. References

[1] See, E.G., A. V. Aho, R. Sethi and J.D. Ullman, *Compilers: Principles, Tools and Techniques* (Addison Wesley Publishing Company, Reading, MA, 1986); R. Sedgewick, *Algorithms* (Addison Wesley Publishing Company, Reading, MA, 1983).

[2] J.V. Noble, "Avoid Decisions", *Computers in Physics* 5:4 (1991) p 386.

[3] J. Basile, "A Forth Finite State Machine", *J. Forth Appl. and Res.* 1:2 (1982) pp 76–78

[4] E. Rawson, "State Sequence Handlers", *J. Forth Appl. and Res.* 3:4 (1986) pp 45–64.

[5] C. E. Eaker, *Forth Dimensions* Vol II No. 3 September/October 1980, pp 37–40.

[6] HS/Forth, Harvard Softworks, P.O. Box 69, Springboro, OH 45066.

[7] ANSI X3.215-1994, *American National Standard for Infomation Systems — Programming Languages — Forth*, American National Standards Institute, New York, NY 1994.

[8] D.W. Berrian, "Forth Based Control of an Ion Implanter", *Proc. 1989 Rochester Forth Conf.*, (Inst. for Applied Forth Res., Inc., Rochester, NY 1989) pp. 1–5.

[9] J. Woehr, *Forth: The New Model* (M&T Books, San Mateo, CA, 1992) p. 43ff.

[10] A.K. Dewdney, *The Turing Omnibus: 61 Excursions in Computer Science* (Computer Science Press, Rockville, MD, 1989), p. 154ff.

[11] J.V. Noble, *Scientific Forth: a modern language for scientific computing* (Mechum Banks Publishing, Ivy, VA 1992). See esp. Ch. 11 and included program disk.

Listing Fourteen

```
3 WIDE FSM: (id)
\ input: |  other  |  letter   |    digit    |
\ state  ------------------------------------
   ( 0 )       NOOP >8     1+ >1       NOOP >2
   ( 1 )       NOOP >8     1+ >2       1+   >2
   ( 2 )       NOOP >8     1+ >3       1+   >3
   ( 3 )       NOOP >8     1+ >4       1+   >4
   ( 4 )       NOOP >8     1+ >5       1+   >5
   ( 5 )       NOOP >8     1+ >6       1+   >6
   ( 6 )       NOOP >8     1+ >7       1+   >7
   ( 7 )       NOOP >8   NOOP >8       NOOP >8  ;

: state<
    [COMPILE]   '   LITERAL  ; IMMEDIATE

\ compile address of "state" cell of a FSM
\ for F83  or ANS replace  '  with  ' >BODY

: <id>   ( $end $beg -- f)                  \ f = T for id, F else
         0  state< (id) !                   \ initialize state to 0
         BEGIN   DUP C@  [ id]   (id)       \ run fsm
                 DDUP  >                    \ $end > $beg ?
                 state<  (id) @  2 <        \ not terminated ?
                 AND                        \ combine flags
         WHILE   1+                         \ $beg = $beg+1
         REPEAT   DDROP                     \ finish loop, clean up
         state< (id)  @   8 <  ;            \ leave flag
```

Listing Fifteen

```
VARIABLE id.len   0 id.len !
: +id.len   id.len  @  1+   id.len  !  ;    \ increment counter
: >1?    id.len @  7 <  DUP  1 AND  SWAP  NOT  2 AND +  ;
( -- 1 if id.len < 7, 2 otherwise)

3 WIDE FSM: (id)
\ input:     |  other  |  letter     |    digit    |
\ state      ------------------------------------
   ( 0 )       NOOP >2   +id.len >1    NOOP >2
   ( 1 )       NOOP >2   +id.len >1?   +id.len >1?  ;

: <id>   ( $end $beg -- f)                  \ f = -1 for id, 0 else
         0 id.len !   0 state< (id) !       \ initialize
         BEGIN   DUP C@  [ id]   (id)       \ run fsm
                 DDUP  >                    \ $end > $beg ?
                 state< (id) @  2 <         \ not terminated ?
                 AND                        \ combine flags
         WHILE   1+  REPEAT   DDROP         \ $beg = $beg+1
         state< (id) @   1 =  ;             \ leave flag
```

Listing Sixteen

```
128 TAB: [ fp#]                          \ decoder for fp#
1 ' [ fp#]  ASCII E  ASCII D  install
1 ' [ fp#]  ASCII e  ASCII d  install
2 ' [ fp#]  ASCII 9  ASCII 0  install
3 ' [ fp#]  ASCII +  +  C!
3 ' [ fp#]  ASCII -  +  C!
4 ' [ fp#]  ASCII .  +  C!

: #err   CRT       \ restore normal output
      ." Not a correctly formed fp#" ABORT   ;  \ fp# error handler

5 WIDE FSM: (fp#)
\ input:     |  other  |   dDeE    |   digit  | + or -  |    dp    |
\ state:     ------------------------------------------------------
   ( 0 )       NOOP >6    NOOP >6     1+  >0    NOOP >6   1+     >1
   ( 1 )       NOOP >6    1+   >2     1+  >1    #err >6   #err   >6
   ( 2 )       NOOP >6    #err >6    NOOP >4    1+   >3   #err   >6
   ( 3 )       NOOP >6    #err >6     1+  >4    #err >6   #err   >6
   ( 4 )       NOOP >6    #err >6     1+  >5    #err >6   #err   >6
   ( 5 )       NOOP >6    #err >6    #err >6    #err >6   #err   >6 ;

: skip-  DUP C@  ASCII - =  -  ;            \ skip a leading -
\ Environmental dependency: assumes "true" is -1

: <fp#>   ( $end $beg -- f)
        0 state< (fp#) !                    \ initialize state
        skip-                               \ ignore leading - sign
    1-  BEGIN   1+  DUP C@  [ fp#]  (fp#)   \ run fsm
                DDUP  <                     \ $end < $beg ?
                state< (fp#) @   6 =  OR    \ terminated by error ?
        UNTIL   DDROP                       \ clean up
        state< (fp#) @   6 <   ;            \ leave flag
```

Listing Seventeen

```
5 WIDE FSM: (fp#)
\ input:     |  other  |   dDeE    |   digit  | + or -  |    dp    |
\ state:     ------------------------------------------------------
   ( 0 )       NOOP >4    NOOP >4   +mant >0?   NOOP >4   1+     >1
   ( 1 )       NOOP >4    ?1+  >2   +mant >1?   #err >4   #err   >4
   ( 2 )       NOOP >4    #err >4   +exp  >3    1+   >3   #err   >4
   ( 3 )       NOOP >4    #err >4   +exp  >3?   #err >4   #err   >4  ;
```

Listing Eighteen

```
4 WIDE FSM: <Fixed.Pt#>
\ input:  |    other?  |   num?     |    minus?   |    dp?     |
\ state:  ------------------------------------------------------
   ( 0 )  | DROP | >0  | EMIT | >1  | EMIT  | >1  |  EMIT  | >2
   ( 1 )  | DROP | >1  | EMIT | >1  | DROP  | >1  |  EMIT  | >2
   ( 2 )  | DROP | >2  | EMIT | >2  | DROP  | >2  |  DROP  | >2  ;FSM

: state<    '  >BODY   LITERAL  ; IMMEDIATE
: Getafix   0   state< <Fixed.Pt#>  !
            BEGIN   KEY   DUP   13 <>      WHILE
            DUP   cat->col#  <Fixed.pt#>   REPEAT ;
```

Writing for Forth Dimensions

I once received an author's manual from another technical magazine. It was 26 pages of intimidating restrictions, rules, and regulations. You got the impression that, even if you managed to learn and obey all those constraints, the editors would be doing you a big favor if they deigned to publish your work.

Our guidelines, by contrast, are meant to encourage you. They are minimal, in order to accommodate different kinds of writing (scholarly/academic, anecdotal, philosophical, editorial, how-to, tutorial, etc.) and so as not to discourage anyone from sharing their Forth knowledge and experience.

Being published in the most widely read Forth periodical brings both personal and prfessional benefits. And the Forth community's vitality springs from good ideas, well communicated, circulating through many interesting and informed minds. We encourage you to write, and we hope that, like our other authors, you will be pleased when your article appears and will want to write often.

If you have an idea but aren't sure if it would make an appropriate article, or if you have any other questions, please feel free to contact me. It will be my pleasure to assist.

—Marlin Ouverson, Editor
editor@forth.org
c/o Forth Dimensions
100 Dolores Street, Suite 183
Carmel, California 93923 USA

What Makes a Good *FD* Article

We'd like to hear your suggestions. If you *feel a subject is important, interesting, befuddling, or helpful, chances are good that many of your fellow readers of* FD *will agree. Here are just a few ideas from our list of editorial wishes:*

- Tutorials about fundamental Forth techniques like factoring, minimizing stack juggling, and readability.
- Description of interesting details of ANS Forth that differ from previous practice.
- Application stories that show Forth in action. Describe the challenges, tell how you arrived at a solution, and share the final results.
- Classical programming problems demonstrated in Forth (e.g., sorts, filters, date routines) and efficient Forth versions of useful features found in other languages (e.g., strings á la Snobol, *grep* from C).
- Human-interest profiles of successful Forth vendors, corporate users, implementors, and innovators.
- Hardware-related pieces about, for example, Forth chips, embedded systems, robotics, and data acquisition. Such articles are most useful if they include "how-to" info.
- Experimental work that charts new directions for Forth or that addresses perceived deficiencies.
- Work that consolidates previously published material or that builds on or subtantially improves earlier articles.
- Ways to improve Forth's visibility and acceptance.
- Academic papers, including Computer Science perspectives, are welcome although *FD* is not formally refereed.

Tips for New Writers — and getting rid of "blocks"

Every writer in an individual, so what encourages one may inhibit another. But if you can't seem to get started—or to finish—one of these ideas might help.

- Plan your article. Try top-down design and bottom-up writing: Start with your subject, make an outline, and check the logical flow of the information. Write the sub-sections, then work on smooth transitions between them. Add a motivational introduction and overview, then write a conclusion with observations, suggestions, and a summary.
- Thinking too critically or analytically while writing can dry you up. If this happens, just relax and jot down all your important points without regard for logic, format, spelling, etc. This is the brain-dump phase, and no one but you will see it. Later, simply rewrite and organize for clarity, focus, conciseness, organization, and completeness.
- Challenge the reader. When you take a stand or issue a challenge, readers tend to get involved, thinking and acting on their own—a worthy goal of many writers. So provoke readers to improve upon or extend your work, and to test it in their own environments; and encourage them to report their findings to *FD* in an article or letter to the editor.
- Have a beta test. Ask a friend or co-worker to read your article, or present it at a local FIG Chapter meeting to see if it communicates as well as you hope and to elicit useful feedback. Accept advice that makes your code or technique or article better (and tactfully ignore the rest).
- Ask for help. Often there are people at work, at the local FIG Chapter, or at on-line Forth venues who are happy to critique your ideas or even to co-author an article.
- Finally, *learn when to let go.* No one ever feels completely ready to deliver that code or manuscript. Like many an application program, an article is subject to endless revision, refinement, and improvement until it ships. If it communicates well enough, is complete enough, and is accurate, ship it and move on!

(Continues on page 31.)

Safer Numeric Input

If, in a 16-bit Forth, you type:

```
1234567890 . <cr>
```

the system responds:

```
722 OK
```

Often, it's not OK. Many will remember that a patient was killed when an X-ray treatment machine delivered a massive overdose. The operator's display had shown the proper dose but, because of attempted corrections to the input, the displayed number was not the internally active number. Ever since I read that report, I have made sure that such an error could never occur in any of my programs. On a CRT with addressable cursor, I erase and rewrite the entire number after each change.

I am now designing a system that uses a 20-key keypad and an LCD display as the operator interface. It is built around an NMIY-0020 board from New Micros Inc., and its 68HC11 processor has a MAXforth kernel in ROM. The keypad and display are standard devices, and the board has built-in interfaces to them.

My intention is that one or the other of these checks ... suffices to guarantee a number on the stack that matches the number on the display.

Keypad code supplied by New Micros uses a table-driven translator to map the keys to ASCII. By switching two pairs of wires, I am able to read the keypad directly as numbers, simplifying some of my code. The keys are labeled 0–F, and four blank keys with removable transparent covers are easy to label as desired. The keys, together with the interface, produce hex codes 0–13; with my alteration, they mean what they say. Codes 10–13 are *minus*, *clear*, *backspace*, and *end*.

The display I use is a passive liquid crystal display, but lighted LCDs and plasma displays that have the same pinout and respond to the same command set are available. The displays accept ASCII characters, have addressable cursors, and can report cursor position and the character at the cursor. They also have enough RAM so that eight characters can be defined by the program. (I can have a °, the degree symbol—nice!)

The code in the listings doesn't erase and rewrite after every change. Rather, the number is built on the display, and is read from the display when the operator presses End. The application must address the cursor to the part of the display where the number is to appear before using `Lcd#` to build the number. `Lcd#` expects TOS to be a number that sets the maximum number of digits, and leaves with a validity flag in TOS, and the number itself below it. It works in any base from binary to hex, and accepts only keys that have meaning in the current base. `Lcd#` won't accept more digits than specified, but a minus sign (accepted only as the first character) doesn't count as a digit.

`Lcd#` can handle 16-bit numbers. The validity flag doesn't check all possible errors. Because the checks appropriate to signed and unsigned numbers differ, and `Lcd#` can input either, either a final check must be made by the calling application, or the code must be further specialized. The variable `invert#` contains the intended sign of the returned number. It has meaning until another number is built. For unsigned numbers, it must be zero (or the user entered a minus sign—you can check); and for signed numbers, it must agree with the actual sign of the returned number. My intention is that one or the other of these checks, combined with the validity flag, suffices to guarantee a number on the stack that matches the number on the display.

There is a way to get a wrong input, but it is easily avoided. The cursor will wrap from the end of the display back to the beginning, but not the other way. An attempt to build a number in a place where it can wrap around the end must be avoided. I can fix the code, but why?

There are many people I must thank for the help they provided in making this code better than I could have made it on my own: Rob Chapman, for the insights he provided into MAXforth; the people at NMI, who produced my nifty little system, supplied it with sample code, and made it a joy to use; and the many participants at comp.lang.forth from whom I learned much simply by reading their posts to others, and who always answered my how-do-I questions.

Additional code

The author can share "a little more code" that might be of interest to someone who wants to put messages on a display (e.g., lcd-type); contact him via e-mail.

Jerry Avins • Kendall Park, New Jersey
jyavins@erols.com

The author held positions at RCA Laboratories' David Sarnoff Research Center, New Brunswick Scientific, and others before retiring in 1987.

```
 ( Listing Zero: words needed to load the rest

: \ 0 WORD DROP ; IMMEDIATE \ Skip-rest-of-line word. (I like to have two.)

: WITHIN ( n lo-limit hi-limit+1 -- ? ) OVER - >R - R> U< ;


\ Listing One

BASE @ HEX

\ Support for 2 line by 40 character display. Thanks with nod to NMI  JYA 2aug98

     B080 CONSTANT lcd-cmd    B081 CONSTANT lcd-data

\ Wait until the LCD isn't busy. (This can hang. Multitasking is better.)
: lcd-wait ( -- ) BEGIN lcd-cmd C@ 80 AND 0= UNTIL ;

\ Send command or data to LCD.
: lcd-cmd! ( C -- ) lcd-wait lcd-cmd C! ;
: lcd-emit ( C -- ) lcd-wait lcd-data C! ; \ LCD analog of EMIT

\ Make specific LCD command words.
: lcd-make ( c -- ) CREATE , DOES> ( -- ) @ lcd-cmd! ;

 1 lcd-make lcd-clear     2 lcd-make lcd-home    8 lcd-make lcd-off
 C lcd-make lcd-on ( On, no cursor )   E lcd-make lcd-cur ( On, with cursor )

: lcd-attr ( blink? cursor? display? -- ) \ Sets three display attributes
      0= 3 + 2* SWAP   0= 1+ OR 2* SWAP   0= 1+ OR   lcd-cmd! ;

\ Initialize the LCD. Specific to a two-line by 40 display.
: lcd-init ( -- ) 01 0C 06 38 38   5 0 DO lcd-cmd! LOOP ;


\ The next 2 words translate cursor locations by number to and from LCD
\ data RAM addresses and add (on write) or strip (on read) the control bits.
\ They too are specific to a two-line by 40 display.

: lcd-at ( a -- ) 80 OR DUP A7 > IF 18 + THEN lcd-cmd! ; \ Write cursor addr
( Undocumented: FF lcd-cmd! resets the cursor, so BS doesn't need to check.)

: lcd-at? ( -- a )  lcd-cmd C@ DUP  27 > IF 18 - THEN ; \ Get current cursor.
\ This supposes that the display isn't reading GC RAM. The supposition is
\ true if cg-addr! hasn't been executed, or if lcd-at was executed after it.
\ (cg-addr! is used to generate special characters in character-generator RAM.
\ That code isn't part of this listing.)

\ Carriage return, line feed, backspace, ->, and <-. All but BS and lcd- wrap.
: lcd-cr ( -- ) lcd-at? 28 / 28 * lcd-at ; \ Beginning of line
: lcd-lf ( -- ) lcd-at? DUP 27 > IF 28 - ELSE 28 + THEN lcd-at ; \ Other line
: lcd-bs ( -- ) lcd-at? 1- DUP lcd-at BL lcd-emit lcd-at ;  \ Destructive BS
: lcd+   ( -- ) lcd-at? 1+ lcd-at ; \ Advances cursor position
: lcd-   ( -- ) lcd-at? 1- lcd-at ; \ Backs cursor position

BASE !
```

\ Listing Two

```
\ Grayhill Keypad Series 86, marked 86JB2-202 G-97-059-J-9742 D$FPJ
\ Leads H <-> J and L <-> M are interchanged to simplify decoding.
\ NMI's Keypad.4th, modified. JYA; 7Aug98 Revised for F.D. 21Sep98

\ KEYPAD ROUTINES

BASE @ HEX

B00A CONSTANT keypad

\ True if a keypad key is pressed. Debouncing doesn't seem to be problem!
: ?keypad ( -- flag )   B000 C@ 1 AND ;

( Wait for keypad key to be released. This can hang at the user's whim!
: kp-release ( -- )   BEGIN ?keypad 0= UNTIL ;

\ wait for a keypad key
: get-key ( -- button ) (
  BEGIN ?keypad UNTIL  keypad C@  2/ 2/ 2/ ;

: num->char ( n -- c ) DUP A < IF 30 ELSE 37 THEN + ;

\ Return the key number, wait for key release )
: kp-key ( -- n )  get-key kp-release ; \ Keypad analog of KEY

( NUMBER INPUT VIA keypad AND DISPLAY

( wait for a single digit. Reject any non-digit and continue waiting
( return the value of the digit 0 to BASE 1-
: 1DIGIT ( -- n ) ( n -> LCD
   BEGIN kp-key DUP BASE @ < NOT WHILE DROP REPEAT
   DUP num->char lcd-emit ;

BASE !
```

\ Listing Three

```
\ A relatively foolproof way to read numbers with a keypad and alphanumeric
\ display. The number is first built on the display (in any base) and then
\ read back onto the data stack. TOS is validity flag which reports overflow
\ and catches non-numeric characters. The application can check twos-
\ complement wrap-around by comparing the sign of the result with the contents
\ of invert# before another number is built. When building the number, only
\ allowed keys are accepted. That is, only numeric keys are effective in base
\ 10, the minus sign works only in the first position, and only as many
\ entries are accepted as the calling the argument specifies. (The editing
\ keys continue to work.)     JYA 9Aug98 Revised for F.D. 21Sep98

BASE @ HEX       0 CONSTANT NO  -1 CONSTANT YES
VARIABLE invert#    VARIABLE #symbols   VARIABLE valid#
: new# ( -- ) NO invert# !    0 #symbols !    YES valid# ! ;
: good-key ( n -- n ? ) \ Flag is false if key need not be dealt with
        DUP 2DUP              ( n n n n )
        10 > SWAP             ( n n ? n )
        BASE @ < OR SWAP        ( n ? n )
        10 = #symbols @ 0= AND
        invert# @ 0= AND OR ;     ( n ? )
```

```
: 1symbol ( -- n ) ( 10 through 13: -, ESC, BS, enter )
    BEGIN kp-key good-key NOT WHILE DROP REPEAT ;

: key-bs ( -- )  \ Remove the last entry
           #symbols @ IF
             lcd-bs  #symbols 1-!
           ELSE
             invert# @ IF
               lcd-bs  NO invert# !
             THEN
           THEN ;

: restart# ( -- ) \ Remove all entries
        #symbols @ invert# @ + ?DUP IF
          0 DO lcd-bs LOOP
        THEN new# ;

: char->num ( c -- n )
           DUP DUP 3A 41 WITHIN SWAP 30 < OR IF NO valid# ! THEN
           DUP 39 > IF 37 ELSE 30 THEN - ;

: build# ( n_max -- n_actual ) new#  lcd-cur \ n is the number of digits.
        BEGIN 1symbol DUP 13 = NOT WHILE
          DUP 10 = IF invert# 1+! 2D lcd-emit THEN
          DUP 11 = IF restart# THEN
          DUP 12 = IF key-bs THEN
          OVER #symbols @ > OVER 10 < AND IF
            #symbols 1+! num->char lcd-emit ELSE DROP
          THEN
        REPEAT 2DROP #symbols @ ;

: read# ( n_actual -- result valid_flag ) \ lcd-on
         0 SWAP DUP IF
           DUP lcd-at? SWAP - lcd-at
            0 DO BASE @ UM* lcd-data C@ char->num DUP
            BASE @ < NOT IF NO valid# ! THEN
            0 D+ IF NO valid# ! THEN LOOP
           invert# @ IF NEGATE THEN
         ELSE NO valid# ! DROP
         THEN valid# @ ;

: Lcd# ( n_max -- value valid_flag ) build# read# ;

BASE !
```

EXPRESS Factory Control

EXPRESS is a programming and operating system for the IBM type of computer, especially suitable for running machinery or running a factory. Years of development by FORTH, Inc. have made it competitive with programs written by large programming companies. Its particular specialty is real-time performance—more difficult than one would think but, of course, well-suited to Forth. An electrician testing my system pressed the switches on my simulator to see how fast the software would follow, pronouncing it successful in contrast to other systems he had seen. An industrial program requires industrial prices to keep it supported, so you won't see EXPRESS in a hobby environment.

My system took four years to develop and is 4.2 Megabytes of compiled Forth, large by EXPRESS standards. It runs almost all of four lime kilns, rock and lime distribution, and a crusher having nothing to do with the rest of the lime plant. Local control is effected with locally placed General Electric Programmable Logic Controllers (PLCs) distributed at the various use sites. A Stargate computer card from FORTH, Inc. provides eight RS-232 ports with eight individual small programs communicating with those ports. The PLCs are used mostly as dumb I/O terminals with the following exceptions: The more-distant PLCs and the complicated ones have a PLC program that toggles a bit back and forth at about 1/2 Hz, said bit to be read by EXPRESS. Another toggling bit is established at the EXPRESS end for each PLC. If either end or both ends fail to receive the other's toggling bit for five seconds, the PLC shuts down the chosen machinery outputs for safety purposes. If shutdown happens, the operator *must* re-establish kiln rotation, by computer or other means, or the lime kiln will be damaged. In real life, most of these errors have taken place by software blunders, especially by yours truly, but we rapidly get the plant operating again without harm. By "we," I mean myself and the electrician assigned to help me. I am not allowed to tamper with software alone while on site.

The superintendent told me to not listen to the workers, so I went out of my way to hear what they had to say...

To help writing the new program, FORTH, Inc. provides an already known good program. MIXER.SRC illustrates good programming techniques, which one can then modify to accomplish the new goal. (FORTH, Inc. also provides training classes to help the starting programmer.)

The way I actually started was to program a minimalist belt system to distribute limestone. The virtues of this approach were that I could practice writing and could introduce the workers to a simple system in order not to overwhelm them. It is important to consider workers' fear, and one should not discount it. At a lime plant hundreds of miles away, the prospect of facing a new, full-blown, running computer system with minimal training was so daunting that several men resigned their jobs. Why? If you wreck a $10,000,000 lime kiln, management will look at you real funny for a long time.

Starting on my end with a simple stone belt distribution system, I drew a screen background with PC Paintbrush in 16 colors. One calls this background into place with the EXPRESS screen editor and then uses other EXPRESS software to lay icons and pushbuttons on this screen. I had a hard time starting up programming and made numerous phone calls to Dennis Ruffer, the main EXPRESS wizard at that time. My main difficulty lay in doing all the steps to make one pushbutton or one digital icon. I have since written my own instructions for doing this.

After getting this program working, we redid it in light of the worker' experiences. The superintendent told me to not listen to the workers, so I went out of my way to hear what they had to say; I mostly got away with implementing their desires. I quickly changed the meaning of the color icons. Now, green is only used for an electric motor running; stopped is gray or dark gray. Red is never used for stop, it means danger, instead. Dark green means that switches are set okay; the opposite is gray, dark gray, or red for tripped fuse. Dark red means a minor danger. This is in contrast to their hard-wired electrical panels, where green may mean a motor is running, but red may mean the same.

In contrast to establishing the icons, programming the logical operations in the `PROCESS`es wasn't so hard. EXPRESS uses a partial infix notation to make the instructions English-like. Also, they have set up the logic as a state machine, which was a new form of logic to me. The idea of a state machine is that if you use a statement to turn an output on, you have to use another statement(s) to turn it off, rather than relying on the negation of the original statement. For example, see Listing One.

In this state machine, the first statement will turn `cs2-LI` on forever without the second statement. It requires the second statement or equivalent statements to provide for turning off `cs2-LI`. The state machine does not scan through the list of rules over and over again like the PLC logic does, for example. The state machine remains idle until an *event* occurs. In this example, the event is the turning on or turning off of `cs2-au` or `cs2-ov`. Then it will scan the list of rules

Allen Anway • Superior, Wisconsin
dwanway@cpinternet.com

Dr. Allen Anway is the world's only freelance EXPRESS programmer.

FORTH INTEREST GROUP
MAIL ORDER FORM

HOW TO ORDER: Complete form on back page and send with payment to the Forth Interest Group. All items have one price. Enter price on order form and calculate shipping & handling based on location and total.

FORTH DIMENSIONS BACK VOLUMES

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

Volume 1 ***Forth Dimensions*** **(1979–80)** **101 – $35**

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

Volume 6 ***Forth Dimensions*** **(1984–85)** **106 – $35**

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

Volume 7 ***Forth Dimensions*** **(1985–86)** **107 – $35**

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

Volume 8 ***Forth Dimensions*** **(1986–87)** **108 – $35**

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

Volume 9 ***Forth Dimensions*** **(1987–88)** **109 – $35**

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 – $35**

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

Volume 11 ***Forth Dimensions*** **(1989–90)** **111 – $35**

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 – $35**

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.

Volume 13 ***Forth Dimensions*** **(1991–92)** **113 – $35**

Volume 14 ***Forth Dimensions*** **(1992–93)** **114 – $35**

Volume 15 ***Forth Dimensions*** **(1993–94)** **115 – $35**

Volume 16 ***Forth Dimensions*** **(1994–95)** **116 – $35**

Volume 17 ***Forth Dimensions*** **(1995–96)** **117 – $35**

NEW **Volume 18** ***Forth Dimensions*** **(1996–97)** **118 – $35**

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 for discussion of technical aspects of applications in Forth. Proceedings are a compilation of the papers and abstracts presented at the annual conference. FORML is part of the Forth Interest Group.

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

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

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

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

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

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

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

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

1993 FORML PROCEEDINGS **323 – $45**
Includes papers from '92 euroForth and '93 euroForth Conferences. Forth in 32-bit protected mode, HDTV format converter, graphing functions, MIPS eForth, umbilical compilation, portable Forth engine, formal specifications of Forth, writing better Forth, Holon – a new way of Forth, FOSM – a Forth string matcher, Logo in Forth, programming productivity. *509 pp.*

1994–1995 FORML PROCEEDINGS (in one volume!) **325 – $50**

Fast service by fax: 408.373.2845

BOOKS ABOUT FORTH

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

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

eFORTH IMPLEMENTATION GUIDE, C.H. Ting **215 – $25**

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

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

Describes the implementation of an 8051 version of Forth. More than half of this book is composed of source listings (w/disks C050) 511 pp.

F83 SOURCE, Henry Laxen & Michael Perry **217 – $20**

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

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

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

F-PC TECHNICAL REFERENCE MANUAL **351 – $30**

A must if you need to know F-PC's inner workings. *269 pp.*

THE FIRST COURSE, C.H. Ting **223 – $25**

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

THE FORTH COURSE, Richard E. Haskell **225 – $25**

This set of 11 lessons 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 the design of embedded software computer systems at Oakland University in Rochester, Michigan. *156 pp.* (w/disk)

FORTH NOTEBOOK, Dr. C.H. Ting **232 – $25**

Good examples and applications – a great learning aid. polyFORTH is the dialect used, but some conversion advice is included. Code is well documented. 286 pp.

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

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

"We're Sure You Wanted To Know..."

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

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

FORTH PROGRAMMERS HANDBOOK, **260 – $57**
Edward K. Conklin and Elizabeth D. Rather

EXCITING NEW TITLE!

This reference book documents all ANS Forth wordsets (with details of more than 250 words), and describes the Forth virtual machine, implementation strategies, the impact of multitasking on program design, Forth assemblers, and coding style recommendations.

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

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

OBJECT-ORIENTED FORTH, Dick Pountain **242 – $37**

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

STARTING FORTH (2nd ed.) Limited Reprint, Leo Brodie **245a – $50**

LIMITED TIME!

In this edition of *Starting Forth*—the most popular and complete introduction to Forth—syntax has been expanded to include the Forth-83 Standard. *(The original printing is now out of stock, but we are making available a special, limited-edition reprint with all the original content.) 346 pp.*

THINKING FORTH, Leo Brodie **255 – $35**

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

WRITE YOUR OWN PROGRAMMING LANGUAGE USING C++,
Norman Smith **270 – $16**

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

WRITING FCODE PROGRAMS **252 – $52**

This manual is for designers of SBus interface cards and other devices that use the FCode interface language. It assumes familiarity with SBus card design requirements and Forth programming. Discusses SBus development for OpenBoot 1.0 and 2.0 systems. *414 pp.*

LEVELS OF MEMBERSHIP

Your standard membership in the Forth Interest Group brings *Forth Dimensions* and participation in FIG's activities—like members-only sections of our web site, discounts, special interest groups, and more. But we hope you will consider joining the growing number of members who choose to show their increased support of FIG's mission and of Forth itself.

Ask about our *special incentives* for corporate and library members, or become an individual benefactor!

Company/Corporate – $125
Library – $125
Benefactor – $125
Standard – $45 (add $15 for non-US delivery)

Forth Interest Group
See contact info on mail-order form, or send e-mail to:
office@forth.org

Fast service by fax: 408.373.2845

DISK LIBRARY
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 designated by the author as public domain, shareware, or use with some restrictions. This library does not contain "For Sale" applications. *To submit your own contributions, send them to the FIG Publications Committee.*

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Version-Replacement Policy

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

★ – Starting ★★ – Intermediate ★★★ – Advanced

MORE ON FORTH ENGINES

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

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

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

Volume 13 (October 1990) **813 – $15**
PALs of the RTX2000 Mini-BEE, EBForth, AZForth, RTX-2101, 8086 eForth, 8051 eForth. *107 pp.*

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

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

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

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

Volume 18 **818 – $20**
MuP21 – programming, demos, eForth *114 pp.*

Volume 19 **819 – $20**
More MuP21 – programming, demos, eForth *135 pp.*

Volume 20 **820 – $20**
More MuP21 – programming, demos, F95, Forth Specific Language Microprocessor Patent 5,070,451 *126 pp.*

Volume 21
MuP21 Kit; My Troubles with This Darn 82C51; CT100 Lab Board; Born to Be Free; Laws of Computing; Traffic Controller and Zen of State Machines; ShBoom Microprocessor; Programmable Fieldbus Controller IX1; Logic Design of a 16-Bit Microprocessor P16 *98 pp.*

MISCELLANEOUS

T-shirt, "May the Forth Be With You" **601 – $18**
(Specify size: Small, Medium, Large, X-Large on order form)
white design on a dark blue shirt or green design on tan shirt.

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

Last 5

DR. DOBB'S JOURNAL back issues

Annual Forth issues, including code for Forth applications.

September 1982, September 1983, Sepember 1984 (3 issues)
425 – $10

FORTH INTEREST GROUP

100 Dolores St., Suite 183 • Carmel, California 93923 • office@forth.org

For credit card orders or customer service:

Phone Orders	*408.37.FORTH*
weekdays	*408.373.6784*
9.00 – 1.30 PST	*408.373.2845 (fax)*

Name _____
Company _____
Street _____ voice _____
City _____ fax _____
State/Prov. _____ Zip _____ e-mail _____
Nation

Non-Post Office deliveries: include special instructions.	The amount of your $ sub-total	The shipping & $ handling
Surface U.S. & International	Up to $40.00	$7.50
	$40.01 to $80.00	$10.00
	$80.01 to $150.00	$15.00
	Above $150.00	10% of Total
International Air		40% of Total
Courier Shipments		$15 + courier costs

Item	Title	Quantity	Unit Price	Total

☐ CHECK ENCLOSED *(payable to: Forth Interest Group)*
☐ VISA/MasterCard:

Card Number _____ exp. date _____

Signature _____

sub-total	
10% Member Discount Member#	
Sales tax* on sub-total *(California only)*	
Shipping and handling *(see chart above)*	
Membership* in the Forth Interest Group ☐ New ☐ Renewal	
TOTAL	

✖ MEMBERSHIP IN THE FORTH INTEREST GROUP

The Forth Interest Group (FIG) is a worldwide, non-profit, member-supported organization with over 1,000 members and 10 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 $45 per year for U.S.A; all other countries $60 per year. This fee includes $39 for *Forth Dimensions*. No sales tax, handling fee, or discount on membership.

When you join, your first issue will arrive in four to six weeks; subsequent issues will be mailed to you every other month as they are published—six issues in all. Your membership entitles you to a 10% discount on publications and functions of FIG. Dues are not deductible as a charitable contribution for U.S. federal income tax purposes, but may be deductible as a business expense.

PAYMENT MUST ACCOMPANY ALL ORDERS

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

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

SHIPPING TIME: Books in stock are shipped within seven days of receipt of the order.
SURFACE DELIVERY:
U.S.: 10 days
other: 30–60 days

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

Fast service by fax: 408.373.2845

Listing One

```
If cs2-au is on  and cs2-ov is on  then  on to cs2-LI ;
If cs2-au is off  or cs2-ov is off then off to cs2-LI ;
```

Listing Two

```
If cf2-bu is on  then off to cf2-bu
   if cf2-au is on  and cf2-ru is on  then off to cf2-ru end
   if cf2-oM is off and cf2-au is on
                    and cm2-ru is on  then  on to cf2-ru end ;
```

Listing Three

```
if ... then ...      ... endif   \  endif:  go to following
if ... then ... else ... endif   \  instruction.
if ... then ...      ... end     \  end:    jump to
if ... then ... else ... end     \  end of the word ;
```

Listing Four

```
PROCESS Example
   BOSS Boss
   MODE State
\  one may define I/O here
\  one may perform logical operations on the I/O
If Boss commands start then  on to State
If Boss commands reset then off to State
       \ 'commands' is synonym for 'is'
END-PROCESS
```

Listing Five

```
\  Process Example,  here I/O detects a fault, so communicate
          ... fault to Boss ;  \  same to superior.
```

Listing Six

```
\  Process Master
If Xexample is fault then ...  \  operate warning system.
```

until all updates have been made, then will remain idle. Not all things are events. A variable known as an INTEGER is not an event, and so cannot be placed before the then because the rule would not fire properly.

FORTH, Inc. has expanded the above structure for more elaborate programming. Consider the rule in Listing Two, taken from an actual program.

The only event required above is cf2-bu being on. In the sub-rules, starting with if, the situations coded are not required to be events; this is especially good for analog situations which are not events. Thus, non-event logic can be written as shown in Listing Three.

A few in the Forth community have found the partial infix notation to be uncomfortable, but I can see its point and I feel at home with it now.

The unit of programming in EXPRESS is the PROCESS. The absolute minimum PROCESS is given in Listing Four.

I have all my processes reporting to the PROCESS Master which must be the head process. FORTH, Inc. has complained that my processes are too few and too big compared with the best programming style, which would include subordinate processes of processes. Processes are intended to be independent and to have minimal communication with each other for good programming strategy. Since this is Forth, one can force communication between independent processes with the TASK command but I use it sparingly, trying to keep good programming practice.

For processes that depend on each other, the usual way to send information between them is to define the communication dictionary APPLICATION-MESSAGES and fill it with whatever words one wants to mention, like fault. The word really has no meaning as we think of it. To the computer, it's just an index number in the dictionary. The subordinate Example will say, as in a logical output statement... [...see Listing Five].

Then one catches this pseudo-command in Master and says something like Listing Six.

Recall that this is a state machine and one must turn all these "commands" off with separate instructions, or else they stay stuck on. Also, I found out that if I invoked rapid-fire logical situations to the Boss from the same subordinate process, the Boss couldn't be relied upon to keep things straight. So I simplified the system by mostly exchanging warning system messages, which are less frequent.

While I programmed the General Electric ladder diagram logic, I was always counting the limited number of timers to be sure I wouldn't run out. With EXPRESS, I have an abundance of these resources, a great convenience.

The biggest advantage is the Forth ability to change the entire system if one wants to. When I first got the system, I made a wish list of how I would like things to be: (a) multiple color icons rather than two colors, (b) choice of decimal-point position in analog readouts, (c) better scales for historical trends, etc. Over the years, I got these features (and so did other EXPRESS programmers) by getting my company to pay a modest fee to FORTH, Inc., and then waiting a week or two for the programming. Changes to non-Forth languages cost ten times more and take a year to accomplish, assuming one can get the vendor to agree to change.

Competition is stiff from other companies, but EXPRESS has true real-time interactive control, which most competition cannot manage. It certainly has worked out well for our application for effective factory control.

Lines and Strings

NEW and OLD Lines

Here is a convenient method for managing two versions of a source file. This may be an existing version and a revision, standard code and a more efficient environmentally dependent version, code you've acquired and your changes to it, etc.

Instead of eliminating or commenting out old lines, prefix them with **[O]** and a space. Prefix your new lines with **[N]** and a space.

The **[N]**-lines will be processed and the **[O]**-lines ignored.

To revert to the old lines in whole or in part, do **Reversion ON** before processing old and new lines. **Reversion OFF** turns that off.

To convert to the new form, use an editor to discard **[N]** and lines prefixed with **[O]**.

```
1 VARIABLE Reversion  ( `Reversion ON` for old lines. )

3 : [N] Reversion @ IF  POSTPONE \  THEN ; IMMEDIATE

5 : [O] Reversion @ 0= IF  POSTPONE \  THEN ; IMMEDIATE
```

Special Characters in Strings

S" ccc" gives us a string literal. But a " can't be in that string. Neither can control characters.

For this purpose, the Tool Belt has **S**. Like other functions in the Tool Belt, **S** takes the first character of the next word as a delimiter.

```
S | Just say "No".| TYPE
```

S also uses **^** before the 32 characters beginning at **@** to yield control characters 0 through 31. **^?** yields 127.

S " Just say^M^J^I^INO^M^J" puts a CRLF (carriage return and linefeed) and two tabs before NO, and CRLF after.

If *ccc* does not contain a " or ^ then **S " ccc"** is equivalent to **S" ccc"**.

```
7 : ?maximum ( n max -- n ) OVER < ABORT" Maximum SizeExceeded " ;

9 ( `str len caddr >control>`  copies a string from str len
10 ( to caddr, converting characters after ^ to control characters.
11 ( It returns caddr and the converted length. )

13 : >control>     ( str len caddr --  caddr k )
14     0 2>R                        ( R: caddr k)
15         BEGIN  DUP WHILE         ( str+i len-i)
16             OVER C@ [ CHAR] ^ CASE? IF
```

Wil Baden • Costa Mesa, California
wilbaden@netcom.com

Wil Baden, after many years of profane language, has retired to Standard Forth. For a copy of the source for this article, send e-mail requesting "Standard Forth Tool Belt #6: Lines and Strings."

```
17                1 /STRING
18                OVER C@  64 XOR
19             THEN                ( . . c)
20             2R@ CHARS + C!      ( str+i len-i)
21             R> 1+  80 ?maximum  >R
22             1 /STRING
23          REPEAT                 2DROP
24      2R>                        ( caddr k)( R: )
25 ;
```

If you know the name of the buffer used by your system for string literals, use it instead of **SBUF**.

```
27 80 CHARS BUFFER: Sbuf

29 : S              ( "<char> ccc<char>" -- caddr k )
30     CHAR PARSE ( caddr k) Sbuf >control>
31     STATE @ IF  POSTPONE SLITERAL  THEN
32 ; IMMEDIATE
```

Appendix

```
: RESERVE ( n -- ) HERE SWAP  DUP ALLOT  ERASE ;
: BUFFER: ( n -- ) CREATE  RESERVE ;

: PARAMETER  BL WORD COUNT EVALUATE ;
: ??  S" IF " EVALUATE  PARAMETER  S" THEN " EVALUATE ;IMMEDIATE

: CASE?   ( x y -- true | x false ) OVER = DUP ?? NIP ;
```


Local Variables for Misers

The following implements the function of local variables with two pre-named variables.

```
2VARIABLE TEMP
MACRO PUSH  " DUP @ >R ! "
MACRO POP   " R> SWAP ! "
MACRO 2PUSH " DUP 2@ 2>R 2! "
MACRO 2POP  " 2R> ROT 2! "
```

At the beginning of your definition: **value TEMP PUSH**.
At the end: **TEMP POP**.

Or, at the beginning of your definition:
***value' value* TEMP 2PUSH**.
value' is in **TEMP CELL+**.
At the end: **TEMP 2POP**.

Within your definition you can use **TEMP** and **TEMP CELL+**.
This recovers and improves the old use of **I** and **I'**. The values in **TEMP** and **TEMP CELL+** are available in called definitions.

Neil Bawd • Goat Hill, California
NeilBawd@Forth.org

Neil Bawd is known for his appearances at FORML. Don't mistake him for his evil twin. His first presentation was "The Ultimate Question" in 1986. For the source for this article send e-mail requesting "The View from Goat Hill: Local Variables for Misers."

Character Tests

In the preceding installment of "Stretching Forth," functions to test characters were given.

```
isalnum  isalpha  iscntrl  isdigit  isgraph
islower  isprint  isspace  isupper  isxdigit
```

In this episode, more functions are given, especially to test characters in a string.

In 1982, KLAUS SCHLEISIEK introduced **SKIP** and **SCAN**. They were employed in the LAXEN-PERRY F83. I used them and **-TRAILING** happily for many years.

However, they are not powerful enough to handle the text I want to massage today. In particular, they consider only a single character at a time, and do not handle a space character as Standard Forth requires.

Take **-TRAILING**. It scans a string from back to front looking for a non-blank. This was fine when source and text were on blocks, but today I want to skip back across control characters as well.

First, let's factor the backward scan process into two parts.

```
1 MACRO BACK|   " BEGIN DUP 0> WHILE 1- 2DUP CHARS + C@ "
2 MACRO |BACK   " 0= UNTIL 1+ THEN "
```

The traditional **-TRAILING** can be defined:

```
: -TRAILING ( str len -- str len-i ) BACK| BL = |BACK ;
```

Now let's define a modern version of **-TRAILING** that will back over blanks and control characters.

```
4 : BL? ( char -- flag ) [CHAR] ! -  94 U< NOT ;

6 : TRIM ( str len -- str len-i ) BACK| BL? |BACK ;
```

BL? is used to identify characters that match a blank in Standard Forth parsing.

TRIM was adopted from Snobol. First I updated **-TRAILING** but there are uses of **-TRAILING** surviving that need the original definition.

SKIP has also been replaced with a pair of macros.

```
8 MACRO SKIP|   " BEGIN DUP WHILE OVER C@ "
9 MACRO |SKIP   " WHILE 1 /STRING REPEAT THEN "
```

Definitions of **SKIP** and **SCAN** can be written.

Wil Baden • Costa Mesa, California
wilbaden@netcom.com

WIL BADEN, after many years of profane language, has retired to Standard Forth. For a copy of the source for this article, send e-mail requesting "Stretching Forth #21: Character Tests."

```
: SKIP ( str len ch -- str+i len-i ) >R SKIP| R@ = |SKIP R>DROP ;
: SCAN ( str len ch -- str+i len-i ) >R SKIP| R@ - |SKIP R>DROP ;
```

Balancing **TRIM**, we often want to skip past blankish characters.

```
13 : JUST ( str len -- str+i len-i ) SKIP| BL? |SKIP ;
```

To parse strings from a longer string, use **SPLIT|** ... **|SPLIT**.

```
15 MACRO SPLIT|  " 2DUP SKIP| "
16 MACRO |SPLIT  " 0= |SKIP DUP >R 2SWAP R> - "

18 : SPLIT-AT-SPACE ( a n -- a+k n-k a k ) SPLIT| BL?|SPLIT ;
```

Forth-type words can be parsed from a line with **JUST** and **SPLIT-AT-SPACE**.

The test functions we have work for common classes of characters. Sometimes we want more or less.

One way is to use the **SKIP|** ... **|SKIP** and **SPLIT|** ... **|SPLIT** macros with an expression in the middle. Thus, suppose we wanted to skip past digits, commas, periods, and dollar signs.

Or suppose we wanted a test that a character is a vowel.

Functions to do this are not difficult, but still are a bother to write.

For problems like these, we want to batch characters in a test. For this we define **[ANY]**.

The stack effect is (*char "<char><space>ccc<char>" — flag*).

The two instances above can be written:

```
SKIP| [ANY] " 0-9,.$" |SKIP

: isvowel ( char -- flag ) [ANY] " AEIOUaeiou" ;
```

The delimiter may be something else than " when we want to test for " in the batch.

The function works by compiling the string into a bitmap. Individual characters or ranges of characters may be given. Control characters can be given by the **^X** convention.

As a character - must be first or last, ^ must be last. - and ^ may end an **x-y** range. Control characters **^X** may not be in ranges.

In the compilation, **set-char-match** converts from a character string to the bitmap. It goes across the string given in the source, character by character, treating each as a control character, range, or single character. Special tests are made on the first and last characters.

set-bit takes a character as a bit offset, and the working bitmap, and sets the appropriate bit in the bitmap.

addr&mask converts a bit offset and a bitmap to an address and mask.

bit&byte takes a bit offset and converts it to a bit and byte offset.

get-bit-as-flag is used to get the result. It takes a character as a bit offset, a bitmap, and the length of the bitmap (which it discards), and returns true when the appropriate bit in the bitmap is set.

SLITERAL compiles the bitmap.

```
21 16 CONSTANT Bit-Map-Size    ( 16 or 32 )

23 Bit-Map-Size CHARS BUFFER: The-Bit-Map

25 ( Convert bit offset to bit&byte offset. )
26 : bit&byte                              ( n -- bit byte )
27    DUP 7 AND  SWAP 3 RSHIFT    ( Could be " 8 /MOD " )
28 ;

30 ( Convert bit offset and string to addr&mask. )
31 : addr&mask                             ( n str -- addr mask)
32    >R  bit&byte CHARS  R> + ( bit addr') 1 ROT LSHIFT
33 ;

35 ( Set bit n in a string. )
36 : set-bit                               ( n str -- )
37    addr&mask ( addr mask) OVER C@ OR  SWAP C!
38 ;

40 ( Get bit n in a string as TRUE|FALSE. )
```

```
41 : get-bit-as-flag                        ( n str len -- flag)
42     DROP ( n str) addr&mask ( addr mask) SWAP C@ AND0<>
43 ;

45 ( As a single char `-` must be first or last, `^` must belast. )
46 ( They may end a `x-y` range. )
47 ( Control chars `^X` may not be in ranges. )

49 : set-char-match      ( str len -- )
50     The-Bit-Map Bit-Map-Size 0 FILL
51     DUP ANDIF OVER C@ [ CHAR] - = THEN ( Take care of leading`-`. )
52          IF  [ CHAR] - The-Bit-Map set-bit  1 /STRING  THEN
53     1- BEGIN  DUP 0> WHILE
54               OVER C@ [ CHAR] ^ = IF       ( Control char )
55                    1 /STRING
56                    OVER C@ 64 XOR
57                    The-Bit-Map set-bit
58               ELSE
59               OVER C@ [ CHAR] - = IF       ( Range x-y )
60                    OVER 1 CHARS - C@ >R
61                    OVER CHAR+ C@ 1+ R>    ( . . hi lo)
62                    2DUP > NOT IF 2DROP ELSE
63                         DO  I The-Bit-Map set-bit  LOOP
64                    THEN                   ( str len)
65                    1 /STRING
66               ELSE                        ( Ordinary )
67                    OVER C@ The-Bit-Map set-bit
68               THEN THEN                   ( str len)
69               1 /STRING
70      REPEAT                               ( str len)
71      DUP 0= IF  OVER C@ The-Bit-Map set-bit  THEN  ( Lastchar. )
72      2DROP
73 ;

75 : [ANY]           ( char " <char> string<char>" -- flag)
76           CHAR PARSE set-char-match                        ( )
77           STATE @ IF
78                 The-Bit-Map Bit-Map-Size  POSTPONE SLITERAL
79                 POSTPONE get-bit-as-flag
80           ELSE
81                 The-Bit-Map Bit-Map-Size get-bit-as-flag
82           THEN                                    ( flag)
83 ; IMMEDIATE
```

Here is how the Standard character test functions *might* have been written.

```
: isupper  [ ANY] " A-Z" ;
: islower  [ ANY] " a-z" ;
: isalpha  [ ANY] " A-Za-z" ;
: isalnum  [ ANY] " A-Za-z0-9" ;
: isdigit  [ ANY] " 0-9" ;
: isxdigit [ ANY] " 0-9A-Fa-f" ;
: isgraph  [ ANY] " !-~" ;
: isprint  [ ANY] "  -~" ;
: iscntrl  [ ANY] "  -~" NOT ;
: isspace  [ ANY] "  ^I^J^K^L^M" ;
```

Forth to the IRS

I *hate* making out tax forms (who doesn't?). When I was still doing my own a few years ago, I decided to ease the sting a bit by combining the task with something I really liked to do—programming Forth. I am showing you what I did as an example of simple system design for a specific purpose. *Note!* The actual code shown here conforms to the U.S. tax code as it was thirteen years ago. There have been numerous changes since, so this code *cannot* be used "as is" to figure your 1999 taxes!

The actual task of developing the system was really one of translation, because the 1040 tax booklet is already a program with a few peculiarities. It is written in English (sort of!) and designed to be executed by humans. Like any program, the goal is to be complete, unambiguous, and correct. All I had to do was select the portions relevant to my situation and translate them into Forth.

Form 1040 can be looked at as a top-down design, a tree structure of calculations and sub-calculations ("Dividends (attach schedule B if over $400)... subtract line 9b from line 9a and enter the result..." etc., etc.). A little thrashing around established a good general pattern for a Forth-based approach:
Rule 1: Use a separate Forth word for each calculation step.
Rule 2: All words take zero arguments and leave one double-precision value on the stack.

This combination made it easy to extend the calculations as necessary in a top-down design (implemented bottom-up, of course). As usual in Forth-based U.S. financial calculations on 16-bit systems, all amounts are in cents and in double precision.

In addition, I found the following utility word useful:

```
: $. (d1 ---)
  \ Print an amount in cents as
  \ dollars and cents
  <# # # # 46 HOLD #S #> TYPE ;
```

Now to some of the system specifics.

(Note to financial voyeurs: the numbers have been changed to protect the innocent.) My wife and I both worked. We lived in a suburb of Detroit, and she worked in the city, so we had to calculate federal, state, and local (Detroit) income taxes. Federal taxes were on both our incomes, Detroit taxes just on hers, at the non-resident rate.

So we set up the following on the income side:

```
32535.71 2CONSTANT GAIL.W2.WAGES
59708.43 2CONSTANT LEN.W2.WAGES
3 CONSTANT EXEMPTIONS
```

Note that this follows the general rule that a word leaves one value on the stack. We are also taking advantage of the Forth convention that numeric input goes to double-precision integers when the number has a decimal point, but its position is ignored, so results will be expressed as an integer number of cents. This is an environmental dependency.

Then we have:

```
: CITY.DEDUCTIONS ( --- d1)
  \ Total city income deductions
  600.00 DROP EXEMPTIONS UM* ;
: CITY.TAX ( --- d1) \ City income tax
GAIL.W2.WAGES CITY.DEDUCTIONS D- 15 1000 M*/ ;
```

The city tax rate for non-residents was 1.5% after deductions.

For the feds, the hierarchy works this way:

```
: TAXES.DUE ( --- d1)
  INCOME.TAX WITHHOLDING D- ;
: WITHHOLDING ( --- d1) 9927.57 3153.05 D+ ;
: INCOME.TAX ( --- d1)
  TAXABLE.INCOME
  47670.00 D- 38 100 M*/ 10171.60 D+ ;
```

This is the Forth translation of "tax owed is $10,171.60 plus 38% of the amount over 47670."

As an example of breaking things down as far as we need to, but not farther, let's look at

```
: DEDUCTIONS ( --- d1)
  TAXES INTEREST D+ CHARITY D+ MISC D+ ;
: MISC ( --- d1)
  DUES BOOKS D+ CONFERENCES D+ ;
: DUES ( --- d1)
  144.00 ( NASW)
  15.00  ( SEMCO)  D+ 56.00 ( AIAA) D+
  96.00  ( ESD)    D+ 56.00 ( ASA)  D+
  50.00  ( TIMS)   D+ 20.0  ( FIG)  D+ ;
```

Here we have an improvement on an adding machine tape; all the amounts are commented to identify them, they are aligned in columns for easy reading when the screen is printed, and if necessary they can be corrected or extended by editing.

That should be enough to give a flavor of the process. The full system is just more of the same, tailored to the specific calculation needed. Each line on the tax form can be arrived at by typing the appropriate word followed by **$.** My estimate is that it took a little longer than it would have by hand. On the other hand there was the personal pleasure of expressing it in code, and the opportunity to correct mistakes or add last-minute items without having to re-do everything. Finally, by printing ten screens, I had a permanent record of exactly how I had done all the calculations. QED (quite easily done). The whole thing is only ten screens, because I only had to code for *my* taxes, not those of the whole world. This brings home Chuck Moore's wisdom when he recommends avoiding over-generalization.

Len Zettel • Dearborn, Michigan
zettel@acm.org

EuroForth 1998 Conference

Getting There

This year, my son Jason and myself were to be traveling companions. Jason is a fairly typical seven year old, with definite likes and dislikes. Introducing him to new food ideas was going to be an interesting challenge. We flew from Bristol Lulsgate Airport on the first leg of our Journey, Brussels and the delights of Belgium. The idea was to look up a few friends from my time at Europay and to do a bit of sightseeing (mainly Brussels and Waterloo).

During our few (albeit wet and windy) days, we managed to see the attractions of the Atomium (a very large model of an iron atom), Autoworld (a historical car museum), the squares and statues of Brussels (including its most famous, The Mannekin Pis). In Waterloo, or rather more south near Braine L'Allieu, we steeped ourselves in the history of 1815—the Battle of Waterloo.

On the Thursday, we left Waterloo for Brussels and thence to Luxembourg. I had planned a stop-over for few hours as a break from a train journey of over six hours. We walked round for a while then spotted a touring Dotto which had a recorded presentation explaining the history of Luxembourg (well worth the fare if you only have a short while). It seems that Luxembourg was built up and destroyed several times over during its more than 1000 year history.

Finally, we boarded our next train, for Trier, getting into a discussion on programming with two Spaniards and an Englishman. Naturally, yours truly espoused the benefits of programming in Forth. At Trier, just time for a quick refreshment then on to Saarbruken by the next train. At Saarbruken, we took the train to Turkismuhle (which turned out to be two, with a change at Saint Wendell) and then a taxi to Schloss Dagstuhl (near Wadern).

The Venue — Schloss Dagstuhl

Schloss Dagstuhl, or Dagstuhl manor house, was built in 1760 by the then-reigning prince Count Anton von Öttingen-Soetern-Hohenbaldern. After the French Revolution and occupation by the French in 1794, Dagstuhl was temporarily in the possession of a Lorraine ironworks. In 1806, the manor house, along with the accompanying lands, was purchased by the French Baron Wilhelm de Lasalle von Louisenthal. In 1959, the House of Lasalle von Louisenthal died out, at which time the manor house was taken over by an order of Franciscan nuns, who set up an old-age home there. In 1989, the Saarland government purchased the manor house for the purpose of setting up the International Conference and Research Center for Computer Science. The first seminar in Dagstuhl took place in August of 1990. Every year, approximately 2,000 research scientists from all over the world attend the 30–35 Dagstuhl Seminars and an equal number of other events hosted at the center.

Internationalisation Workshop

The Internationalisation (i18n) Workshop preceded the main conference, occupying the Friday morning. The need for this was the result of the ANS to ISO Fast-Track vote when the European Forth Community declared a need for internationalisation (i18n) issues to be considered in the amendments to the ANS document. I18n issues cover not only language and cultural aspects, but also time and date formatting. There was a concern that any effort in i18n should not break existing code. This ruled out changing the character-access words `C@`, `C!`, `CMOVE`, and `COUNT`. This led to the proposal that a new Internationalisation Wordset be created. Some of the discussion related to the adequacy or otherwise of UNICODE characters. Whilst it is true that the most commercially significant languages are catered for, there are still a number of languages and character sets that are not. Many of these are variable-length characters. The workshop ended by agreeing on the basic wording of a proposal to the ANS Technical Committee.

There is an Internet mailing list which is in operation to continue discussion of the specific proposals, and anyone who needs to make their views known on this issue should join the discussion. This is run by Anton Ertl and you can subscribe by sending e-mail to:

anton@mips.complang.tuwein.ac.at

The Conference

The conference proper began in the afternoon. The first session was on "Forth Philosophy and Standardisation" and was kicked off with a paper by M.L. Gassanenko but presented by Sergei Baranof. Russia and its economic problems meant that Sergei's ticket had to be organised from outside Russia, leaving Sergei to represent Russia in its entirety (or as much as he was able). The subject of the first paper was "Context-Oriented Programming."

Egmont Woitzel then gave us "Transient Scope, Word Lists and Language Construction." This compared the state-driven method and prefix-driven methods. In the state-driven method, the search order is made implicit by state transitions. In prefix-driven methods, there needs to be a consideration of the search order as it is effected by the modularity strategy, parse strategy, and the transient scope strategy. The paper identified the need for improvements in interface and naming conventions and source code management. Header suppression was seen as a complicating matter for maintenance and debugging. The audience felt that this was more about scoping control for personal use rather than wider area of usage.

Peter Knaggs then took the floor to explain the structure and form of the continuing ANS (and subsequently ISO) standardisation efforts. It seems that Forth was the first "ex-

Paul E. Bennett
peb@transcontech.co.uk

Paul E. Bennett is the Systems Engineering Director of Transport Control Technology Ltd., who develop High Integrity Distributed Embedded Control Systems (HIDECS) and certified Forth-based software.

tensible" language to be standardised. He briefly covered Forth's problem (extensibility) and a solution to gaining standardisation via an open pre-standardisation review. This revolved around gaining a wide review audience for new words and wordlists proposed for standardisation. There is, therefore, a new procedure proposed which is under consideration by the TC.

Peter went on to inform us about the changes in ANS committee structures. The ANS Technical Committees are now ITI Technical Committees (even though the standards publications remain ANS). NCITS (which was the X3 secretariat) now incorporates J14 (the Forth Sub-Committee). The result of all this is that the committee has a new designation and is now known as NCITS/J14.

The first ANS standard was "fast-tracked" to an ISO standard, and ANS have the responsibility for its maintenance. However, when the ISO standard was issued, there was a proviso that the Internationalisation Issues be looked at for the next revision. The ANS standard is now up for revision, and the following changes are expected to be made:

a) All words marked as obsolescent in the current standard to be deleted.
b) Ratification of all clarifications issued in response to the nine requests already answered and the two requests that remain outstanding.
c) Support for embedded and ROMable systems.
d) Support for internationalisation and extended character sets.

Items c) and d) are, of course, both demands from the ISO fast-track process.

Updating the ANS standard is a two-year process.

Anyone can join the TC if they fulfill the membership criteria. This includes paying $300 per year and attending the first two meetings (in the USA) and two thirds of all meetings. You also have to vote in more than 80% of all letter (e-mail) ballots. Meetings are July and November.

The NCITS/J14 committee can set up task groups to look at specific items and produce a report. Membership to this is open to anyone for $300 per year (TC members get on for free).

Finally, it was proposed that a public notice board be established as part of the NCITS/J14 web site:

file://ftp.uu.net/vendor/minerva/uathena.htm

This would enable announcements and posting of the new proposed words and wordsets to be aired for comment.

JOHNATHON MORRISH of Micros Engineering managed to do a paperless paper on the PIC entitled "Spiders and Forth." This featured the device in various sizes of configuration and packaging, and mentioned some real applications (including one involving "Cats-Eyes," the devices that divide the lanes on UK roads). NICK NELSON, of the same company, also did a paperless paper, "A Grid Control in Forth," which dealt with graphical objects in windows environments. REUBEN THOMAS expounded on "Mite: a Fast and Flexible Virtual Machine," while SERGEI BARANOF covered again for M.L. GASSANENKO and presented "Open Interpreter: Portability of Return Stack Manipulations."

Within session four, on old and new words, ANTON ERTL attacked state-smart words and proposed some interesting solutions, comparing various other system implementations in the process. We also had MICHAEL MILENDORF talking about the proper use of CATCH and THROW, and MANFRED MAHLOW and KLAUS SCHLEISIEK spoke about PRELUDE and FINALE, context-switching words which apply limited pre- and post-execution facilities.

The final session kicked off with the double act of PAUL BENNETT and MALCOLM BUGLER relating the experience gained in applying software certification to a Forth-based medical "life-support" product. The fact that such certification was considered late in the project and was completed in three weeks was partly due to the use of Forth and to the effort and dedication of Paul's team of code reviewers. The certification documentation was accepted by the notified body who were evaluating the product for CE marking under the Medical Devices Directive and further work towards FDA approval. The process did identify some serious flaws in the code which were put right by the authors prior to the CE marking evaluation, and it was recertified. BERND PAYSAN presented another astounding demonstration of MINOS and the way 3-D graphics could be integrated. The final paper of the conference was presented by STEPHEN PELC, who presented a "Portable Forth Optimising Native Code Compiler."

The Conference Circuit

EuroForth is settling down to a three-year cycle. The cycle has a year in the UK (various locations), a year in Germany (mostly at Schloss Dagstuhl), and a year in a guest country. As the 1997 conference was in Oxford, England, and this year was at Schloss Dagstuhl, next year is the turn of a guest country. St. Petersburg has been proposed for 1999, with Vienna in reserve. A final decision will be made in February 1999. This leads the venue back to the UK for the year 2000 and, as this is going to be a significant year for locations meridian, the organisers have decided to pursue the possibilities that might be afforded by Cambridge.

In conjunction with the above, it was also decided that a EuroForth mailing list be established to assist in the organisation (by e-mail) of future conferences and open discussion of conference topics. An announcement has been made in the newsgroups and by e-mail to this year's participants already. For others wishing to subscribe, an e-mail to euroforth-subscribe@egroups.com will get you on. An archive of the mailing list will be available from:

http://www.egroups.com/lists/euroforth/

All this is linked from Peter Knaggs' Forth pages at:
http://www.dec.bournemouth.ac.uk/Forth/

Impromptu Papers

MALCOLM BUGLER — "EPP, the forgotten standard," discussed probably the most useful port on the PC, the parallel port. Having presented a little history, Malcolm went on to espouse the merits of using the port. He also displayed some code for interfacing to the port in EPP mode.

REUBEN THOMAS — "Demi Doubles," a short exploration of double-cell values and words that utilise them.

KLAUS SCHLEISIEK — "Uses of other ASCII codes," really a request for opinions on the standard uses of some of the ASCII punctuations. He was looking for one or two characters for which there was not an already common usage.

PETER KNAGGS — "Journal of Forth Application and Research (JFAR) Status." This report on the current status of the JFAR publication was an update of the report given at last year's conference. Peter has now managed to install himself in the

position of editor and has begun to publish accepted papers on the web. There are plans to print paper copies of the material when enough are collected to make it worthwhile, and to publish the collected papers on CD also. The back-issue paper publications will be fully indexed on-line and may also be included on CD when they are scanned in. The web sites are at:

http://dec.bournemouth.ac.uk/Forth/JFAR/ (main site)
http://www.jfar.org/ (mirror of main site)

PETER KNAGGS — "USA Conferences," mainly a quick talk about The Washington Forth Party which was tentatively booked for the 23rd and 28th July in Washington DC.

HOWERD OAKFORD — "Angles, Trig and Complex Numbers," mainly a plea for help. Howerd expounded the problems he was having considering the more complex maths involved in resolving trigonometric calculations from a complex number input. He felt that keeping the complexors as a pair of cells made sense until the last possible moment before usage. Several offers were made to share code with him.

The Conference Workshops

There were three workshop stream topics this year:

Methods and Objects

This workshop reported that there now seems to be a convergence of ideas and techniques amongst those who are interested in object-oriented wordsets. It seems like a bit more discussion and work may actually get a concensus on this subject. We may even be able to look forward to an OO wordset being proposed for consideration by NCITS/J14.

Project Management

This discussion centred on three aspects.

"Project Start," which identified customers as belonging to two types: those with and those without a specification. This seemed to imply that each type would be best served by different system development models, these being "waterfall" and "spiral," respectively. It was agreed that the waterfall model of development had been written up and explained often enough to be clear to everyone. However, there seemed to be scant information on the spiral model, and it was only alluded to in many reference books without being fully described in project terms.

For a spiral development model, it was agreed that risk assessment, awareness of the customer's overall business goals, and early review of applicable standards and legislation. The client contact needs to be offered the means to "buy-in" to a proposal it to the rest of his company. It was also identified as important to do a "post-mortem" on each and every project to see if there are lessons to be learnt.

"Administration" identified a need for good support tools for version management, problem reporting and tracking and, in essence, good configuration management. Reviews were seen as necessary to ensure compliance of code to "house" coding and documentation styles, technical integrity of the product, and general quality. A reminder was issued that documentation errors are also bugs, and English documentation is needed with the code to explain the code to non-programmers who may be part of the review team.

"Group Working"—even in Forth, programmers are working in teams. This requires that projects be properly structured, in step with the structure of the product. This helps in team management and provides stronger componentisation.

Virtual Machines

Since Forth is frequently implemented using a virtual stack machine, the available implementation techniques have been studied in some detail over the years. This workshop showed, however, that a number of new questions are being raised in this area. These involve the deployment of Forth virtual machines in complex environments involving Windows and networking, and in particular we discussed issues raised by client-server programming.

"Implementation." Could a stack-based virtual machine be a basis for just-in-time compilation? It was noted that advanced optimisation techniques for converting Forth into native code are being implemented in the latest MPE Forth compilers. Similar techniques could be used for JIT conversion from token or threaded code. Current work on both stack- and register-based virtual machines was discussed. It was noted that Forth, unlike Java, retains a trap door to the underlying physical machine through the ability to add code definitions. This allows it to be tuned very effectively to particular applications.

"Functionality and Portability." The virtual machine needs extending to provide access to GUI toolkits (e.g., TK) and communication technologies (e.g., TCP/IP) at an appropriate level of abstraction.

"Client-Server Deployment." Forth will perform well in a client-server environment, particularly where extended functionality is required. Distributed functionality can be obtained by communicating Forth source text between nodes, possibly in tokenised form. Security issues need to be investigated.

The Competition

It has long been a tradition at Forth conferences to have a fun competition (which may or may not include coding). This year, it was decided that we should relate Forth to everyday life. The task was to write some code that must be:

a) Readable
b) Portable
c) Maintainable

The participants had to work with a well-known phrase containing three previously defined Forth words and three undefined words. The participants had to provide:

1) Name for the definition
2) Stack comment
3) A description
4) Definitions for undefined words

The well known phrase that was provided as a starter was `SEX, DRUGS, ROCK AND ROLL.`

As is usual with this kind of competition, extra points were obtainable for imaginative use of commas, and all attempts to influence the judges are, naturally, gratefully received.

Sadly, not many entries made it by the deadline. The entered codings were all highly amusing, despite some entrants displaying the fact that they had not fully understood the specifications. After due deliberation by the contest judging panel, PAUL BRUIN won through to receive the coveted "Fat Cigar" award, a prize deemed fitting the content of the competition.

One entry that arrived by e-mail, but which did not receive a prize, was from a Mr. W. Clinton, of Little Rock, Arkansas.

```
DEFER DRUGS      \ n ----- ; context sensitive
: BILL'S_DRUGS   \ n ----- ; attitude
    INHALE? NOT ME!
    THROW
;
: SEX            \ --- u; none of your business
   PRIVACY CONTEXT !
   JOURNALIST @ EXECUTE
   MONICA @
;
: ROCK           \ ---
   SHAKE  RATTLE  ROLL
;
: LIFE           \ attitude -- report
   IS DRUGS
   BEGIN
       SEX DRUGS ROCK AND
       TERM TIME-OUT?
   UNTIL
   ROLL OVER
;
BILL'S_DRUGS LIFE
```

Though this was considered useable as a model, it had

Listing One — the winning entry

```
1 CONSTANT XY     \ Male object (constant is best used)
0 CONSTANT YY     \ Female object (variable may be better)

: WAY2LIVE  ( no_fixed_address\XY\YY -- )
 \ Forth proposal for a better ANS standard (of living)
 \ Note: ANS = another non-standard standard
   SEX,            \ Having fun without spawning child processes
   DRUGS,          \ an unconventional (u14l) technique
   ROCK            \ sorting and fine-tuning
   AND             \ consolidation of resources
   ROLL            \ prioritising your life
   !               \ and keeping the results
;

: SEX   ( nfa\XY\YY --nfa\XY )
  \ Having fun without spawning child processes
   2 PICK @        \ if there's anyone at home....
   IF 1 PICK 0     \ we'll try and pick her up
      ?DO          \ then check if she does!
         OVER      \ if successful, do it over..
         OVER      \ and over again (if possible)
         +         \ add it all together
         IF        \ if we got a result
           LEAVE   \ ... we'll make a fast exit
         THEN
      LOOP
   THEN
   DROP         \ if she's still around at this stage, we'll drop her
;
: DRUGS,  ( nfa\XY --- NFA\YX )
  \ trying to scramble/unscramble the data
   OVER >R      \ hold on to the important stuff...
   R@ !         \ before we change the current state
   R@  W@       \ if we aren't completely fulfilled...
   R@ 2+ W@     \ we should try all our options
   SWAP         \ now we must priorotise things
   R@ W!        \ try to hide some stash here
   R@ 2+ W!     \ ..and hide the rest over here
   R> @         \ finally, we try to find ourselves
;
: ROCK  (nfa\YX --- nfa\XY\XY\XY )
   \ sort, stabilise and fine-tune the data
   DRUGS,       \ start off with some more of this
   1 PICK       \ now try some string manipulation (m12n)
   OVER 4*      \ use some clever cellular tricks
   ERASE        \ and prepare our bass (sorry, should be base)
   DUP          \ first make a back-up copy....
   DUP          \ before making a master copy
;
```

some serious problems in the coding (obviously not tested). Another entry, which unfortunately arrived too late for inclusion, but which did work well, was by MALCOLM BUGLER. This entry is not included in this report but may be available for download from the EuroForth web-site with the electronic version of this report.

MICHAEL MILENDORF gave us:

```
HEX       \ A       hex value
          \ BABE    hex value
          \ FACE    hex value

: SEX  ( -- ) RECURSIVE
     SEARCH A BABE
     FIND IF
             SEE FACE AND EVALUATE ,
             PICK BABE , EMIT WORDS ,
             DO ROCK AND ROLL LOOP ,
             DO DRUGS LOOP ,
             BEGIN
                FORGET WORDS
                MOVE AND ROLL WITHIN OPEN SPACE
                THROW
             UNTIL
          ELSE
             SEX
          THEN
;
: ROCK ( -- )
    BEGIN
        MOVE AND ROLL !
        DUMP AND THROW !
        DROP
    UNTIL
;
: DRUGS ( -- )
    BOTTLE PICK AND OPEN
    BEGIN
        REFILL GLASS
        DRUNK
    UNTIL
    BOTTLE DUMP
;
```

This, it was felt by the judges, had missed some of the essence of the competition rules. However, the winning entry presented in Listing One had more merit and was accepted in the obvious spirit of universal harmony that it displayed (or perhaps the author likes having sex with aliens).

Obviously, you need to use the backwards loader to get this one into the computer, as it was written top-down fashion.

Other Prizes

Other prizes were awarded for the best papers and presentation, and a "best dressed for dinner" prize:

ANTON ERTL, for a paper displaying a most-incisive applicability.

PAUL BENNETT and MALCOLM BUGLER, for best real-life application of Forth

JASON BENNETT, best dressed for dinner (stealing the title from Malcolm Bugler).

The Survivors Party

After dinner, Malcolm Bugler organised us into teams of four, passed round the words of a song and tasked us to perform it as a barbershop quartet. After a little practice, the teams gave their often-amusing renditions. There followed a few more impromptu songs and plenty more boozing.

Author Recognition Program

To recognize and reward authors of Forth-related articles, the Forth Interest Group (FIG) has adopted the following Author Recognition Program.

Articles

The author of any Forth-related article published in a periodical or in the proceedings of a non-Forth conference is awarded one year's membership in the Forth Interest Group, subject to these conditions:

a. The membership awarded is for the membership year following the one during which the article was published.
b. Only one membership per person is awarded in any year, regardless of the number of articles the person published in that year.
c. The article's length must be one page or more in the magazine in which it appeared.
d. The author must submit the printed article (photocopies are accepted) to the Forth Interest Group, including identification of the magazine and issue in which it appeared, within sixty days of publication. In return, the author will be sent a coupon good for the following year's membership.
e. If the original article was published in a language other than English, the article must be accompanied by an Engish translation or summary.

Letters to the Editor

Letters to the editor are, in effect, short articles, and so deserve recognition. The author of a Forth-related letter to an editor published in any magazine except *Forth Dimensions* is awarded $10 credit toward FIG membership dues, subject to these conditions:

a. The credit applies only to membership dues for the membership year following the one in which the letter was published.
b. The maximum award in any year to one person will not exceed the full cost of the FIG membership dues for the following year.
c. The author must submit to the Forth Interest Group a photocopy of the printed letter, including identification of the magazine and issue in which it appeared, within sixty days of publication. A coupon worth $10 toward the following year's membership will then be sent to the author.
d. If the original letter was published in a language other than English, the letter must be accompanied by an English translation or summary.

By 11:30 (a.m.) EST (USA) on Tuesday 11 August 1998, the JForth download site had been deluged with hundreds of download requests, overwhelming the system. The system went down for eight hours.

Despite that outage, in the 24-hour period ending midnight of the 11th *(the first day)*, an incredible 1,649 hits had been recorded on the primary site. Auditing is not available for the secondary, ftp site.

In the 21 days after the JForth release on 10 August 1998, the primary download site at www.softsynth.com/jforth had 931 downloads of JForth. The temporary ftp site did not have tracking information (so the actual number is probably a lot higher) and is now closed.

Following the release of JForth freeware on the 10th August 1998, we are now proud to announce the availability of the JForth web site at home.tampabay.rr.com/jforth/.

JForth is an implementation of Forth designed specifically for the Commodore Amiga. JForth uses a 32-bit stack and compiles directly to 68000 machine code. This makes JForth faster than most Forths. JForth also provides an extensive set of tools for accessing the special features of the Amiga. You can call any Amiga Library routine by name and reference any Amiga structure using constructs similar to those of C. JForth also has some special toolboxes that support simple graphics, Intuition menus, IFF files, and other Amiga features. These toolboxes can be used directly to simplify Amiga programming. Source code for these toolboxes is provided so you can customize them or study them as examples of Amiga programming. JForth also provides over a dozen small sample programs for those, like me, who learn best by example.

JForth also allows you to do things that are unique in the Forthexperience, the most dramatic being CLONE. This exceptional utility allows you to create a totally independent, standalone version of your program of minimal size.

Clone will take a compiled JForth program, extract out all of the code and data needed to run it, and re-assemble a smaller version of it. All of the JForth development tools, the name fields and link fields and any other unused words are left behind. The final image size is comparable to images created using a C compiler and linker. Images can be saved with a symbol table for use with WACK or other debuggers (if needed). The JForth Source Level Debugger can also be used with Cloned programs. Most programs will Clone without modification if they follow a few simple rules regarding run-time initialization of variables or arrays containing Forth addresses.

(Continued from page 13.)

What to Include

- ❑ Your name as you wish it to appear in print.
- ❑ Your city, state (or province), and nation of residence.
- ❑ Brief autobiographical information to share with readers—such as education, relevant employment, your introduction to and use of Forth, current projects, and other interests.
- ❑ Your e-mail address for publication, so interested readers can contact you. If you have a personal Web site, you can include that URL as well.
- ❑ If your article depends on a specific Forth dialect or implementation, be sure to specify it (e.g., ANS Forth, Forth-83,
- ❑ Your complete mailing address, so we can mail complimentary contributor's copies of the issue in which your work appears.
- ❑ Permission to post your code, in the form you have sent it to us, on one or more FTP sites so readers can download it; and/or include the URL of an FTP site where you will post the code.
- ❑ Daytime and evening telephone numbers and a fax number (if any). These will *not* be published, and will be used only if we have questions or problems during the editing process.
- ❑ If you send hard-copy manuscript, artwork, or disks that need to be returned to you, include a self-addressed envelope with adequate postage affixed to it.

How to Submit Your Work

If in doubt, just ask—we can work out the details.

E-mail

The fastest, most convenient way for us to receive your material is via e-mail to the editor@forth.org address. Binary (e.g., formatted text) files must be uuencoded to be sent as e-mail, but ASCII files can be sent as-is.

"Snail" mail

Or, mail a hard copy *with* PC or Mac diskettes containing your material to *FD* Editor, c/o the Forth Interest Group.

File formats

If your article includes equations or other items highly dependent on your particular software and/or fonts, include a PostScript file, if submitting electronically, so we can generate a reliable hard copy for proofing. Formatted files that can be read by MS Word are acceptable (e.g., RTF, WordPerfect, and many others); or just send the text as ASCII—you can indicate any critical formatting with informal tags, e.g., <italic>like HTML<end-italic>.

Illustrations

Figures are black-and-white and can be vector (PostScript) or bitmapped. They likely will be re-drawn for clarity, style, and compatibility.

Two Problems in ANS Forth
(with ANS solutions)

Introduction

I doubt that anyone is happy with all of ANS Forth and that any programmer could not list off three or four things which they would class as significant losses or missed opportunities when ANS is compared to F83. Recently, when I suggested on comp.lang.forth that the biggest "bug" in ANS Forth is the presence of colon-sys on the control-flow (and, therefore, potentially on the data stack), I was exaggerating the problem slightly; I really think it's the second biggest problem, but I had already found a workaround for the biggest problem, which, in my eyes anyway, is the inability of the programmer to assign the input stream to an arbitrary block of memory.

The ensuing, rather vigorous, response to my assertion on the newsgroup happily led to Elko Tchernev offering a workaround for the colon-sys problem, which he attributed to Wil Baden and so I am now in the fortuitous position of having a solution for my pet ANS-hates. I still don't like the locals wordset, but at least I don't need to use it.

I present below a short discussion of how these two questions affect the programmer, a general guide to how to solve them, and finally an example of code which gets around them both and does something useful in its own right; viz., it implements a finite state machine wordset which is then used to build an FSM which translates ASCII text into Plain TeX format.

The Problems

1. The problem with colon-sys is that the programmer has no obvious way of determining whether it exists and, if it does, how big it is. This makes it difficult to take into account, to say the least. The colon-sys problem really comes to the fore in applications where a special form of colon is needed but there is no need to have a special semi-colon. The natural word to turn to at these times is :NONAME but this has the stack diagram (-- xt colon-sys). If we want the "special colon" to build a word which uses that *xt*, we have to get it out from under a stack item we do not know the size of. Generally, the solution I have used up to now is to make a special semi-colon to match the new colon word, which performs the normal semi-colon action first, to remove colon-sys, and then does whatever is required to *xt*. This solution is ugly and annoying.

2. The input stream words such as PARSE, WORD, and >IN seem, at first glance, a gift to the programmer who wants to write a simple parser for text files of various sorts. First impressions are deceiving, however. Since there is no ANS-sanctioned method of setting the input stream without transferring control to the (supposedly) Forth words contained in it (i.e., EVALUATE), it is often the case that the function of these words is re-written by the programmer into words which take an input stream definition *c-addr length* pair. This is a waste of programmer effort and is plain inelegant when the code manifestly exists in the Forth kernel already. Indeed, just to rub salt into the wound, the "missing" word we are looking for must be a factor of EVALUATE.

The Solutions

1. The size of colon-sys is, in fact, easily determined by a program as it compiles:

```
: SOMEWORD [ DEPTH ] LITERAL ... ;
```

Since we know from the ANS Forth document that the only thing added to the stack by colon is a colon-sys (or nothing at all, if the control stack is separate), then the depth reported above is the size, on the data stack, of a colon-sys, assuming that nothing was placed on the stack by the program before hand. This allows ROLL or PICK to be used to retrieve items that lie under the colon-sys:

```
: GET-XT [ DEPTH ] LITERAL ROLL ;

: REPORTXT: ( <NAME> -- colon-sys )
  :NONAME  GET-XT DUP ." New xt is: " .
  CREATE ,
 DOES>
  @ EXECUTE
;
```

This allows the new word REPORTXT to be completed by the normal semi-colon. This method is used below to allow the definition of the FSM's state's actions after the state names themselves are defined, making it much easier for states to reference each other.

2. This one is harder and less attractive. To allow a word to use the normal parsing words in an arbitrary region of text, we first ALLOT or ALLOCATE a region of memory big enough for the word which is going to perform the action, plus a space character, plus the length of the text to be parsed. Then we copy in the name of the word, a space character, and finally the text to be parsed. The start address of this buffer is passed, along with its total size, to EVALUATE and hey, presto! the word's actions are applied to the text following it in memory.

The example FSM below uses this method to allow the FSM to parse a file, replacing any characters which would give TeX trouble with the required control sequences in the output file.

A better solution for everyone would be to expose the code in EVALUATE which redirects the input source to the passed string. Call it something like >SOURCE or >INPUT or just READ.

Thomas Worthington • Newcastle Upon Tyne, England, UK
T.W.Worthington@ncl.ac.uk
www.ncl.ac.uk/~n6388131/home.htm

The author's most visible Forth achievement is Aztec Forth for Win32 machines. He has programmed for 20 years, and currently is in the final year of a Software Engineering Degree at Newcastle University.

The Example

I recently downloaded a very large text from the Gutenburg Project which I wanted to be able to treat as a TeX document. However, some ASCII characters, such as "{" and "}" and "&", are treated as commands by TeX and, additionally, the normal ASCII double quote mark " is better as either " or " in TeX, depending on whether it represents an opening quote mark or a closing quote mark.

Clearly, this was a job for a finite state machine which would scan through the text, outputting most text as it appeared in the original but replacing special characters with their TeX equivalents. It would have two states, expecting the next double quote to be an open or a close, and passing to the other state when a quote mark was encountered.

First, I build a set of words for defining and running an FSM:

FSM	A variable which holds the current state of the machine, *as an xt*. (I couldn't very well call it STATE, could I?).
GO	Simply executes the next state until the DONE or ERROR state is arrived at.
STATES	(n <name*1*> ... <name*n*> --) Predefines the names of all the states so that they can refer to each other without constant recourse to DEFER and IS.
(STATE)	Simply defines the action of one of the above states, i.e., it sets the current value of FSM to the state's *xt*.
DONE	Transfers the machine to a final, accepting state.
ERROR	Transfers the machine to a final, non-accepting state.
STATE:	(<name> -- colon-sys) Starts the definition of a state's action, where the state has already been declared by STATES.

This is the code for these words:

```
VARIABLE FSM

: GO \ simply read the current state and if it
     \ is not DONE or ERROR perform its
     \ action; repeat.
  BEGIN
     FSM @ DUP 0>
   WHILE
     EXECUTE
  REPEAT
  DROP
;

: DONE  0 FSM ! ;
: ERROR -1 FSM ! ;

\ a reference to a state causes a transition
\ to that state on the next loop through GO
: (STATE) DOES> @ FSM ! ;

\ Declare your states before you start.
: STATES ( N <NAME1> <NAME2> .. <NAMEN> -- )
   0 ?DO CREATE -1 , (STATE) LOOP ;

\ This uses the colon-sys trick discussed in
\ the text to set up the state's action without
\ having to make a special form of semi-colon.
: STATE: ( <NAME> -- )
  [ DEPTH ] LITERAL ( COLONSIZE)
  >R :NONAME ( XT COLONSYS |r: COLONSZ)
  R> ROLL ( COLONSYS XT <NAME> -- )
  ' >BODY !
;
```

And that's it! These words are enough to declare any FSM—so let's use them.

ASCII>TeX. Needs the file words and the memory allocation words.

```
: >=  < INVERT ;

\ move to next character in the input stream.
: +CHAR ( -- ) 1 >IN +! ;

\ get the next character in the input stream,
\ unless we have run out of stream.
: CHAR> ( -- CHAR TRUE | FALSE)
  SOURCE >IN @ >= IF
    >IN @ + C@   -1
  ELSE DROP 0
  THEN
;

\ for keeping the handle of the output file
VARIABLE OUTFILE

\ write a string to the output file.
: >OUT ( ADDR LEN -- ) OUTFILE @ WRITE-FILE
  ABORT" Write fail!" ;

\ declare our states
2 STATES  NORMAL "WAIT

\ This is the name of the word which will do
\ the work, it is the FSM word GO
: PARSE$ ( -- CADDR LEN) S" GO " ;

\ This ungainly word makes a dynamic buffer for
\ the input file plus the above string,
\ reads in the input file, closes it (in case
\ we are going to over-write it.), places
\ the action word at the start of the buffer
\ (where we have left room for it) and
\ returns the address of the buffer and the
\ total length.
: GETINPUT ( HANDLE -- ADDR LEN)
  DUP >R FILE-SIZE OR
    ABORT" File read problem" ( SIZE|r: HNDL)
  PARSE$ NIP + DUP ALLOCATE
    ABORT" Out of memory." ( SZ ADDR|H)
  TUCK PARSE$ NIP + SWAP R@ READ-FILE
    ABORT" Read failed!"
( ADDR LEN1 |R:HND)
  R> CLOSE-FILE DROP
  PARSE$ ( ADDR LEN1 CADDR LEN2)
  3 PICK SWAP MOVE  \ PUT GO into start of file
  PARSE$ NIP +      \ add length of PARSE$ to
                    \ file length
```

```
;

\ Converts a character from ASCII to TeX if
\ required, otherwise just passes it through.
: FILTER ( CHAR -- )
  CASE
    [CHAR] \ OF  S" $\backslash$"  >OUT  ENDOF
    [CHAR] { OF  S" $\{ $"  >OUT    ENDOF
    [CHAR] } OF  S" $\} $"  >OUT    ENDOF
    [CHAR] $ OF  S" \$"     >OUT    ENDOF
    [CHAR] & OF  S" \&"     >OUT    ENDOF
    [CHAR] # OF  S" \#"     >OUT    ENDOF
    [CHAR] % OF  S" \%"     >OUT    ENDOF
    [CHAR] _ OF  S" \_"     >OUT    ENDOF
    [CHAR] ^ OF  S" \^"     >OUT    ENDOF
    [CHAR] / OF  S" \slash "      >OUT    ENDOF
        DUP HERE C! HERE 1     >OUT
  ENDCASE
;

\ allows input of a string in
\ "words and spaces" form.
: "WORD" ( "<text>" -- caddr length)
  [CHAR] " PARSE 2DROP [CHAR] " PARSE ;

\ load the input file, open the output file,
\ run the FSM, and close everything afterward.
: CONVERT ( "<nameFrom>" "<nameTo>" -- )
  "WORD" R/O OPEN-FILE
      ABORT" File not found."
  >R "WORD" W/O CREATE-FILE
      ABORT" Can't open output file"
  OUTFILE !
  R> GETINPUT ( ADDR LEN)
  OVER >R       \ keep for FREE
  NORMAL EVALUATE   \ Start FSM in NORMAL
  R> FREE
      ABORT" Error releasing memory."
  OUTFILE @ CLOSE-FILE
      ABORT" Error closing output file."
  FSM @ 0<
      ABORT" WARNING: Quotes not balanced!"
;

\ Finally, the two states: NORMAL waits for an
\ open quote, "WAIT waits for a close quote.
STATE: NORMAL
  CHAR>
  IF ( CHAR)
    DUP [CHAR] " =
    IF  DROP S" ``" >OUT +CHAR  "WAIT
      ELSE  FILTER +CHAR
    THEN
  ELSE  DONE  THEN
;

STATE: "WAIT
  CHAR>
  IF ( CHAR)
    DUP [CHAR] " =
    IF  DROP S" ''" >OUT +CHAR NORMAL
      ELSE  FILTER +CHAR
    THEN
  ELSE  SOURCE 1+ >IN ! DROP
  \ Scrap rest of input
        ERROR   THEN
;
```

Once compiled, the program is used by typing

```
CONVERT "filename1" "filename2"
```

where *filename1* is the original ASCII file and *filename2* is the new TeX file to be generated.

The current version does not filter the characters ^ and _ or < and > as these rarely appear in the sort of text which I wrote the program to convert.

SPONSORS & BENEFACTORS

The following are corporate sponsors and individual benefactors whose generous donations are helping, beyond the basic membership levels, to further the work of *Forth Dimensions* and the Forth Interest Group. For information about participating in this program, please contact the FIG office (office@forth.org).

Corporate Sponsors

AM Research, Inc. specializes in Embedded Control applications using the language Forth. Over 75 microcontrollers are supported in three families, 8051, 6811 and 8xC16x with both hardware and software. We supply development packages, do applications and turnkey manufacturing.

Clarity Development, Inc. (http://www.clarity-dev.com) provides consulting, project management, systems integration, training, and seminars. We specialize in intranet applications of Object technologies, and also provide project auditing services aimed at venture capitalists who need to protect their investments. Many of our systems have employed compact Forth-like engines to implement run-time logic.

Computer Solutions, Ltd. (COMSOL to its friends) is Europe's premier supplier of embedded microprocessor development tools. Users and developers for 18 years, COMSOL pioneered Forth under operating systems, and developed the groundbreaking chipFORTH hot/target environment. Our consultancy projects range from single chip to one system with 7000 linked processors. www.computer-solutions.co.uk.

Digalog Corp. (www.digalog.com) has supplied control and instrumentation hardware and software products, systems, and services for the automotive and aerospace testing industry for over 20 years. The real-time software for these products is Forth based. Digalog has offices in Ventura CA, Detroit MI, Chicago IL, Richmond VA, and Brighton UK.

Forth Engineering has collected Forth experience since 1980. We now concentrate on research and evolution of the Forth principle of programming and provide Holon, a new generation of Forth cross-development systems. Forth Engineering, Meggen/Lucerne, Switzerland – http://www.holonforth.com.

FORTH, Inc. has provided high-performance software and services for real-time applications since 1973. Today, companies in banking, aerospace, and embedded systems use our powerful Forth systems for Windows, DOS, Macs, and micro-controllers. Current developments include token-based architectures, (e.g., Open Firmware, Europay's Open Terminal Architecture), advanced cross-compilers, and industrial control systems.

The iTV Corporation is a vertically integrated computer company developing low-cost components and information appliances for the consumer marketplace. iTVc supports the Forth development community. The iTVc processor instruction set is based on Forth primitives, and most development tools, system, and application code are written in Forth.

Keycorp (www.keycorp.com.au) develops innovative hardware and software solutions for electronic transactions and banking systems, and smart cards including GSM Subscriber Identification Modules (SIMs). Keycorp is also a leading developer of multi-application smart card operating systems such as the Forth-based OSSCA and MULTOS.

www.kernelforth.com

An interactive programming environment for writing Windows NT and Windows 95 kernel mode device drivers in Forth.

www.theforthsource.com

Silicon Composers (web site address www.silcomp.com) sells single-board computers using the 16-bit RXT 2000 and the 32-bit SC32 Forth chips for standalone, PC plug-in, and VME-based operation. Each SBC comes with Forth development software. Our SBCs are designed for use in embedded control, data acquisition, and computation-intense control applications.

T-Recursive Technology specializes in contract development of hardware and software for embedded microprocessor systems. From concept, through hardware design, prototyping, and software implementation, "doing more with less" is our goal. We also develop tools for the embedded marketplace and, on occasion, special-purpose software where "small" and "fast" are crucial.

Tateno Dennou, Inc. was founded in 1989, and is located in Ome-city Tokyo. Our business is consulting, developing, and reselling products by importing from the U.S.A. Our main field is DSP and high-speed digital.

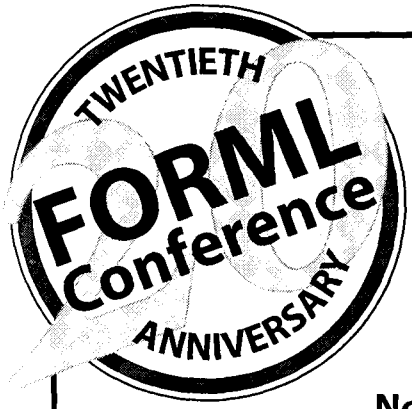
ASO Bldg., 5-955 Baigo, Ome,Tokyo 198-0063 Japan
+81-428-77-7000 • Fax: +81-428-77-7002
http://www.dsp-tdi.com • E-mail: sales@dsp-tdi.com

Taygeta Scientific Incorporated specializes in scientific software: data analysis, distributed and parallel software design, and signal processing. TSI also has expertise in embedded systems, TCP/IP protocols and custom applications, WWW and FTP services, and robotics. Taygeta Scientific Incoporated • 1340 Munras Avenue, Suite 314 • Monterey, CA 93940 • 408-641-0645, fax 408-641-0647 • http://www.taygeta.com

Triangle Digital Services Ltd.—Manufacturer of Industrial Embedded Forth Computers, we offer solutions to low-power, portable data logging, CAN and control applications. Optimised performance, yet ever-increasing functionality of our 16-bit TDS2020 computer and add-on boards offer versatility. Exceptional hardware and software support to developers make us the choice of the professional.

Individual Benefactors

Makoto Akaishi
Everett F. Carter, Jr.
Edward W. Falat
Michael Frain
Guy Grotke
John D. Hall
Guy Kelly
Zvie Liberman
Marty McGowan
Gary S. Nemeth
Marlin Ouverson
John Phillips
Thomas A. Scally
Werner Thie
Richard C. Wagner



Twentieth Anniversary of the FORML Conference

"Forth Interfaces to the World"

November 20–22, 1998 • Pacific Grove, California

FORML welcomes papers on a variety of Forth-related topics, even those which do not adhere strictly to the published theme. Papers submitted at press time include the following (see www.forth.org/Papers.htm for updates):

Color Forth 98, and
A New Command-Line Interface, Charles Moore

Source to RTF,
Object-Oriented Programming in Forth Made Simple, and
The Full Tool-Belt, Wil Baden

OOP and Multitasking USER Variables, Dian Blew

Reconfigurable Forth Processor, John Hart

Forth Philosophy 1998, Glen Haydon

Asynchronous Serial I/O with the PSC1000, Bradford J. Rodriguez, Ph.D.

The 'ZTAR' MIDI Controller, Bradford J. Rodriguez, Ph.D., and Harvey Starr

Open Network Forth -- Control system for the Munich accelerator facility, Ludwig Rohrer and Heinz Schnitter

Forth Multiprocessing, Dr. C.H. Ting

An ANS Forth Target Compiler, John Rible

Report: the ANS Forth Organizational Meeting, Elizabeth D. Rather

Internationalisation—the User Perspective, Stephen Pelc, Willem Botha, Nick Nelson, Peter J. Knaggs

Issues in International Programming, and **Typing Forth**, Peter J. Knaggs, Ph.D.

Object Forth Wraps C Structures, John Sadler

Registration Information

Advance registration required. FIG members are eligible for a 10% discount on any registration fee.

Inquiries about conference registration may be directed to office@forth.org or to FORML Conference Registration, c/o Forth Interest Group, 100 Dolores Street, Suite 183, Carmel, California 93923.

Conference attendee in double room	$595*
Non-conference guest in same room	$435*
Under 18 years old in same room	$225*
Conference attendee in single room	$795*

Infants under two years in same room—free

Conference Chairman: Marlin Ouverson – editor@forth.org
Conference Director: Robert Reiling – ami@best.com

The FORML Conference is held at the Asilomar Conference Center, a National Historic Landmark noted for its wooded grounds just yards from Pacific Ocean dunes and tidepools on California's Monterey Peninsula.

**Lodging and all meals are included with conference registration*, and spouses and guests of conference participants can join numerous recreational outings and activities.

Please confirm your attendance early—accommodations may be limited due to this facility's immense popularity.

Call for Papers

Please submit the subject of your paper as soon as possible in order to be included in pre-conference publicity. Completed papers should be received by November 1 in order to be included in the conference notebooks that are distributed to all attendees. (Late titles and papers will be accepted.)

E-mail submissions may be sent to editor@forth.org with "FORML paper" in the subject line. Hard copy may be mailed to FORML Conference Chairman, c/o Forth Interest Group, 100 Dolores Street, Suite 183, Carmel, California 93923.