FORTH Dimensions

Volume 8, Number 6

March/April 1987
$5.00

Bresenham Line-Drawing

7776 Limericks

DOS File Disk I/O

Inverse Video and TI-FORTH

State of the Standard

Checksum More

Well, what I really want is . . .

a CMOS computer system for dedicated applications
Guess it's time to get the design team going.

Forth Dimensions
Published by the
Forth Interest Group
Volume VIII, Number 6
March/April 1987
Editor
Marlin Ouverson
Advertising Manager
Kent Safford
Production
Cynthia Lawson Berglund
Typesetting
LARC Computing

Forth Dimensions solicits editorial material, comments and letters. No responsibility is assumed for accuracy of submissions. Unless noted otherwise, material published by the Forth Interest Group is in the public domain. Such material may be reproduced with credit given to the author and to the Forth Interest Group.

Subscription to *Forth Dimensions* is free with membership in the Forth Interest Group at $30 per year ($43 foreign air). For membership, change of address and to submit items for publication, the address is: Forth Interest Group, P.O. Box 8231, San Jose, California 95155. Administrative offices and advertising sales: 408-277-0668.

Symbol Table

Simple; introductory tutorials and simple applications of Forth.

Intermediate; articles and code for more complex applications, and tutorials on generally difficult topics.

Advanced; requiring study and a thorough understanding of Forth.

Code and examples conform to Forth-83 standard.

Code and examples conform to Forth-79 standard.

Code and examples conform to fig-FORTH.

Deals with new proposals and modifications to standard Forth systems.

FORTH Dimensions

FEATURES

DEPARTMENTS

Letters to the Editor

Words, Required & Controlled

Dear FIG,

Glen Haydon's essay, "A Forth Standard?" (*FD* VIII/4) is a good discussion of some of the difficulties of standardizing an extensible, evolving, community-designed language such as Forth. Too little standardization makes it difficult to port applications between implementations and discourages newcomers from learning Forth. Too much standardization stifles evolution of the language. His solution makes sense: don't make a word part of the official standard until it has become accepted into common usage.

Ideally, adherence to a standard allows any application written on one vendor's system to run on any other standard system, but this ideal is seldom met. Programs written in one vendor's C, Pascal or BASIC will often not run without modification on a different implementation. I agree with Mr. Haydon that requiring each Forth application to be written entirely in the Required Word Set and standard extensions is unnecessarily restrictive. Few programmers will be willing to give up other useful words found in their implementation just for portability. On the other hand, I think everyone will agree that standard words should not have different meanings in different implementations.

An important but underemphasized part of the Forth-83 Standard is the Controlled Reference Set — words which, although not required, cannot be present with a non-standard definition. Here is the appropriate place for most of Mr. Haydon's common-usage words. Reserve the Required Word Set for those words that no implementation can do without. Don't try to expand the Required Word Set to make it self-sufficient. To do so would require too much conformity from vendors, forcing them to leave out innovative variations which might otherwise never find their way into common usage and thence into some future standard. A case can even be made for moving some of the Required Word Set into the Controlled Reference Set, especially words which have more to do with the implementation than the language itself, such as file words. Would a system be any less Forth-like if it used named, random-access files instead of blocks?

Standards are important, but they can also be overdone. Now that there is talk of an ANSI standard for Forth, it would be good if the Forth community could come to some concensus about what shape they would like it to take, or even if such a standard is a good idea at all.

Sincerely,

David Nye
Eau Claire, Wisconsin

Test Your Assertions

Some method of testing assertions has become a common feature in many programming languages (See, for example, *ACM SIGPLAN Notices*, August 1976, pp. 36–37). An example is the "assert" statement in most versions of C. I have used this Forth version for some time and have found it invaluable.

The rationale is as follows: in writing a program with a complicated logic structure, a point is often reached where we think, "If the program gets to this point and there are no bugs in my logic, then this and this must be true." **[ASSERT]** is intended to assure ourselves that "this and this" really are true. This anti-bugging should be done as the code is written and the conditions occur to you. This simple and natural mechanism has many advantages over temporary, *ad hoc* debugging tools, particularly in documentation and the fact that the assertions are transparent as maintenance changes are made, unless an error occurs.

To use the **[ASSERT]**, execute whatever sequence of words you assume to be true, ending with a Boolean flag on

```
Screen # 33
( Compile time part of assert       NLH  1/27/87 )   \  [ASSERT]
( Assert tf on stack top: if not, show scr.#, line #, & call- )
( ing addr, else clean up stack.  )              (  ---  )
: [ASSERT]   ?LOADING  ?COMP    ( Only valid if compiling from  )
      BLK @     ( Block number being LOADed.                disk. )
      B/SCR     ( Number of blocks per 1K screen.                 )
      /MOD      ( Quotient will become screen number & remainder )
                ( will become line number in screen after 2 *.   )
      16 *  SWAP  2 *    ( Shift quotient to left 3 half bytes;  )
                       ( therefore, max screen number is OFFFh.  )
      IN @  64  /   ( Count into text buffer divided by len. of  )
                    ( lines on a scr.; after + becomes line no.  )
      +  +      ( Combine screen no. & line no. into one number. )
      [COMPILE]  LITERAL     ( Compile number for run-time use.  )
      COMPILE  (ASSERT)      ( Compile CFA of run-time assert.   )
      ;  IMMEDIATE

Screen # 34
( Run-time part of assert            NLH  12/8/86 )   \  (ASSERT)

          ( flag[ from application ]  u[ from [ASSERT] ]  --- )
          (  ^^^^                      ^                       )
: (ASSERT)      SWAP     ( Application flag determines action.   )
        IF  DROP     ( If flag is true, then assertion is true & )
                   ( no action is taken except to clean up stack. )
        ELSE CR  16  /MOD  ( Separate pieces of assert literal.  )
           ." ASSERT Scr # "   .        ." Line # "    .
           R>  R  2-  H.  >R   CR    ( Show the addr. within the )
                ( word calling the word containing the [ASSERT]; )
              ( calling word may be an applic. word or INTERPRET. )
           KEY  ASCII C  -     ( True if entry is not a "C".     )
          IF  QUIT  ( If not wish to "C"ontinue, QUIT - leaving )
          THEN         ( the stack intact for further debugging. )
        THEN   ;
```

Hills Screens

the data stack. This may be by a **DUP** of essential processing. Then write **[ASSERT]**. During execution, if the flag is true, the flag is dropped and execution continues; otherwise, the screen number and line number from which the **[ASSERT]** was compiled are displayed, along with the address of the call to the word containing the assertion. (This address allows you to determine where you came from in case of multiple calls.) Then, entry of a "C" allows execution to resume, while any other key results in a **QUIT**, which preserves the data stack.

The word definitions are essentially fig-FORTH. **ASCII C** may be replaced by 67, and **H.** may be replaced by **U.** or defined as

```
: H.   BASE @   SWAP
  HEX   U.   BASE  !  ;
```

If the assertions *must* be removed from a production version, redefine **[ASSERT]** as **DROP**. May all your assertions be true!

Norman L. Hills
Des Moines, Iowa

Out of Ireland

Dear Mr. Ouverson,

After reading reference to Henry Laxen's multi-tasking tutorial, and being very interested in giving my fig-FORTH implementation multi-tasking capability, I felt compelled to purchase the relevant back issues of *Forth Dimensions* (V/4,5).

The tutorial taught me a great deal, but being a novice programmer, I was unable to convert Henry's 8080 example to run on my 6502. I think this should be possible, using a BRK instruction instead of RST. I would be very interested in hearing from any of your readers who might have performed the conversion.

I am involved in hardware design, and at present I am working on a product using voice processing. I find Forth a very convenient language from the hardware point of view, in that I can generate test code very quickly and without the tedium of assembly coding.

I would be interested in reading more about real-life commercial applications of Forth in the pages of *Forth Dimensions* in the future. I'm sure Forth is doing many interesting things out there!

Sincerely,

Richard Rooney
Kilsallaghan
Co. Dublin, Ireland

F1...F5 for F83

Dear Marlin,

Well, I've finally done it! After I mistyped a line five times in a row, I decided it was time to make the DOS editing keys (F1 - F5, etc.) available in Forth (Laxen and Perry's F83 for IBM PC and compatibles only).

The basic trick is to redefine **EXPECT** to make use of the DOS string-input function. It's simple, but it has side effects. The original **EXPECT** did its own key decoding. The new one, shown in Figure One, leaves it to DOS to do so. (**EXPECT** is defined on screen 49 — all numbers in decimal — of KERNEL86.BLK.)

This now allows the use of the DOS editing keys to edit the string last entered. It also allows the use of the Superkey command stack to play back any of a number of recent entries. ("Superkey," from Borland Int'l. Software, is a memory-resident "keyboard enhancer" program that allows pre-defined key sequences, etc. It also keeps track of the last twenty or so entries keyed in on the command line for DOS and DEBUG.)

Now, about those side effects.

The original **EXPECT** uses a translation table (**CC-FORTH**) to handle various control characters. That will no longer work with this version of **EXPECT**. The definitions of screens 47 - 48 of KERNEL86.BLK are used only through **EXPECT**. They do not function with this new definition of **EXPECT** and are no longer required. If you do bypass them, also disable the references in screens 91, line 9; and screen 92, line 3.

Primarily affected are control-C (reset), control-P (toggle printer) and control-X (delete line). Also affected is the handling of CR.

To accept control-C, add the two lines in Figure Two (lifted from Wil Baden's F83X) between the existing lines 4 and 5 of the unnamed cold start routine on screen 85 of KERNEL86.BLK. This changes the BIOS break vector so that on control-C or control-break, Forth is restarted.

To toggle the printer, use the Forth words **PRINT ON** and **PRINT OFF**.

To delete the line being entered, use the escape key rather than control-X.

A more insidious problem is that this DOS function echoes all characters to the screen, including the CR character that terminated the input. (In normal Forth mode, that CR displays as a single blank. Now, any subsequent output would overwrite the displayed input. The Forth word **CR** is included in this **EXPECT** to clean up the display. It does cause unnecessary blank lines. It also makes the opening **CR** in words such as **WORDS** and **DIR** superfluous.

You'll note that I'm using a separate internal buffer for this **EXPECT**. The DOS convention dictates that the first two bytes of the buffer are for maximum length and length actually used. Since this differs from the way Forth uses its buffers, it could possibly affect some other words using **EXPECT**. An internal buffer of adequate size avoids that problem.

A last side effect may or may not be a problem. The original **EXPECT** will exit on a buffer-full condition. This **EXPECT** will exit only on a CR. This may affect certain applications. If so, keep both versions around.

Note that the second count/length byte controls what the DOS editing keys "remember." You can actually change the buffer content (string), and make sure that the second length byte agrees with the length of the string.

That is, make sure:

```
(LIT)  D  EXPBUF
1+  COUNT  +  C!
```

For those who haven't done this before, recompile your Forth system using the following steps. Make sure the files listed below, as well as those included through screen 1 of EXTEND86.BLK, are on the disk(s) you're using. After editing the source screens of KERNEL86.BLK, type the following lines in turn:

```
OPEN META86.BLK
1 LOAD
BYE
KERNEL EXTEND86.BLK
OK
BYE
F83 PDE.BLK
```

(or **FSED.BLK** or **VED.BLK**, etc.)

```
1 LOAD
SAVE-SYSTEM F.COM
```

```
VARIABLE EXPBUF 128 allot    \ tot len 130
: EXPECT    (S  addr  len  ->  )
  EXPBUF C!  EXPBUF  10 BDOS  DROP  CR
  EXPBUF 1+ COUNT  DUP SPAN !  ROT SWAP MOVE ;
```

Van Duinen — Figure One

```
AX AX SUB AX DS MOV 140 # BX MOV 256 # AX MOV   \ Set Brkpt to
AX 0 [BX] MOV    CS AX MOV    AX 2 [BX] MOV      \ restart pgm
```

Van Duinen — Figure Two

```
ASSEMBLER DEFINITIONS
: FOO   ( -- ) HERE . ;       \ A boring example

ALSO FORTH DEFINITIONS
SYNONYM FOO FOO

FOO
```

Bradley — Figure One

```
...
IF      DOES> @ EXECUTE
ELSE    DOES> STATE @
        If   @ ,
...
```

Bradley — Figure Two

```
: DO-IMMED ( -- )  DOES> @ EXECUTE  ;   ( for an immediate synonym )
: DO-COMP  ( -- )  DOES> @  STATE @     ( for a non-immediate synonym )
   IF  ,    ELSE  EXECUTE  THEN
;
: SYNONYM    ( -- )  \ new-name old-word
   CREATE                                ( make header for new word )
   HIDE  BL WORD FIND DUP  REVEAL        ( old word found? )
   IF    SWAP ,                          ( yes, compile its cfa )

         IMMEDIATE                       ( make new word immediate )
         1+                              ( was old word immediate? )
         IF    DO-EXEC                   ( yes, set new to execute )
         ELSE  DO-COMP                   ( no, set new to check state )
         THEN
   ELSE  1 ABORT" not found"             ( old word not found )
   THEN
;
```

Bradley — Figure Three

(Those last four lines only if you are using a full-screen editor such as one of these.)

Frans Van Duinen
Toronto, Ontario

An Alias for Synonyms

Dear Marlin,

The synonym technique described by Victor Yngve (*FD* VII/3) is very useful. I have been using the same technique (albeit under a different name; I called it **ALIAS**) for about two years, and find it to be an important part of my toolkit.

There is a subtle bug in the published implementation. Fortunately, the bug is easy to fix. Suppose you have a word **FOO** in, for example, the **ASSEMBLER** vocabulary. You wish to use **SYNONYM** to make that word also appear in the **FORTH** vocabulary. So you try the code in Figure One and — BOOM! — the system crashes. What happened? Well, the name **FOO** is now a synonym for itself, not a synonym for the word named **FOO** in the **ASSEMBLER** vocabulary.

The solution is the same as the solution used for preventing recursion when redefining a colon definition: **HIDE** the word just after the **CREATE** and **REVEAL** it after **FIND** (systems other than F83 probably need **SMUDGE** ... **SMUDGE** rather than **HIDE** ... **REVEAL**).

I also have a minor implementation quibble. The sequence shown in Figure Two is unstructured in the sense that the child word effectively jumps into the middle of the **SYNONYM** word (at @ **EXECUTE**) and then depends on the **ELSE** in the **SYNONYM** word to keep it from executing the rest of the definition after the **EXECUTE**. This could cause problems in some implementations. Figure Three shows a structured version, including the **HIDE** ... **REVEAL** fix.

Thanks to Victor Yngve for a fine article!

Mitch Bradley
Mountain View, California

```
Screen # 70
( Usage: SYNONYM <new-name> <old-name>   Forth-83 r8/27/86 vhy)

: SYNONYM      ( -- )
    CREATE                          ( make header for new word )
    32 WORD FIND DUP                ( old word found? )
    IF SWAP ,                       ( yes, compile its cfa )
       IMMEDIATE                    ( make new word immediate )
       1+                           ( was old word immediate? )
       IF   DOES> @ EXECUTE         ( yes, set new to execute )
       ELSE DOES> @ STATE @         ( no, set new to check state )
            IF   ,                  ( and compile if compiling )
            ELSE EXECUTE            ( or execute if executing )
            THEN
       THEN
    ELSE 1 ABORT" not found"        ( old word not found )
    THEN  ;

Screen # 71
( SYNONYM  Glossary entry                     revised 8/27/86 vhy)

SYNONYM            --
  A defining word used in the form:
        SYNONYM <new-name> <old-name>
  Create a dictionary entry for <new-name> so that when
  <new-name> is later used, it will have substantially the same
  action that <old-name> would have had.  If <old-name> was
  immediate, the action will be immediate, otherwise not.
  During compilation, the action is to compile the same thing
  that <old-name> would have compiled.  The dictionary search
  order is not changed and <new-name> must be different from
  <old-name>.
```

Yngve Screens

Mr. Yngve replies:

Thanks to Mitch Bradley for raising some interesting points and for noticing that a @ can be factored out, thus shortening the definition. The improved version is given on screen 70.

We call it a bug when a program does not do what we expect it to. We can fix it by changing the program or by changing our expectations. We could say that **SYNONYM** is restricted to not directly redefining a word by the same name, using instead the pattern

```
SYNONYM FOO A
SYNONYM A FOO
```

Or we could use a colon definition if there are problems with vocabularies. This change in expectations is reflected in the new glossary entry on screen 71. It retains the advantage that **SYNONYM** is still defined completely within the Forth-83 Standard, so it will work without change on any standard system. It has the disadvantage that **SYNONYM** is not completely general purpose.

To make it completely general, one would not only have to include the relevant version of **HIDE** ... **REVEAL** or **SMUDGE**, but also implementation-specific code such as **CURRENT @ CONTEXT !** to change the search order, and perhaps other non-standard code as well. This would have the disadvantage of making **SYNONYM** difficult to install on a different standard system or to transport a program containing it to a different system. Nevertheless, this was the option taken in **MACRO** ... **END-MACRO** (*FD* VII/3) so as to make it similar to : ... ; in its use. Ideally, the facilities needed for programming general versions of such words should be added to the standard.

In regard to the minor implementation quibble, the given definition seems to me to be perfectly legal according to the Forth-83 Standard. The suggested alternative requires defining and naming three words instead of one and it ties up more overhead in dictionary space.

Forth treats both **:** ... ; and **IF** ... **ELSE** ... **THEN** as structures subject to error checking when the word containing them is compiled. But the sequence **CREATE** ... **DOES>** is no more a structure than **CREATE** ... **ALLOT**, for **CREATE** can be used alone without **DOES>** or **ALLOT**, and **CREATE** can be replaced by a word containing **CREATE**. Any word containing **DOES>**, however, is unstructured in a different sense: the first part (before **DOES>**) executes when the word is used to compile a child, and the second part (after **DOES>**) executes when the child is run. The **DOES>** exits from the word after compiling into the child a jump back into the word. This is the very essence of **DOES>**, and the source of its power. Thus in

```
: ... DOES> ... ;
```

the nesting code compiled by the colon executes at the child's compile time but the unnesting code compiled by the semicolon executes at the child's run time! Just as there is no problem using **DOES>** to make a child jump into the middle of a colon definition, provided there is a proper return on the return stack, there is even less of a problem using **DOES>** to make a child jump into the middle of an **IF** ... **ELSE** ... **THEN** construction. It's OK to jump in between **ELSE** and **THEN** because no code is compiled by **THEN**. It's OK to jump in between **IF** and **ELSE** because **ELSE** simply compiles an unconditional branch around the code between **ELSE** and **THEN**, and this unconditional branch at run time is in no way dependent on the preceding conditional branch compiled by **IF**.

I cannot think of any possible implementation of the standard where the given definition of **SYNONYM** would not work and the suggested alternative

(Continued on page 31.)

Bresenham Line-Drawing Algorithm

Phil Koopman, Jr.
North Kingstown, Rhode Island

The task of drawing a straight line on a graphics screen is a fundamental building block for most computer graphics applications. Unfortunately, this capability is not included in many Forth implementations and, for that matter, is not included in the ROM support programs for many personal computers. This article will show you how to draw lines on almost any graphics display, and gives complete listings in MVP-FORTH.

The CRT Display Layout

First, let's establish some conventions. I will assume that the graphics display on your computer is addressed using (X,Y) Cartesian coordinate pairs, where X and Y are both non-negative integers (see Figure One). The point (0,0) — also called the origin — is the upper-left corner of the computer screen. Each addressable point on the screen is called a pixel (short for "picture element"). The X coordinates represent columns of pixels (horizontal distance from the origin), and the Y coordinates represent rows of pixels (vertical distance from the origin).

The exact number of pixels on your computer's display screen is hardware-dependent. However, some representative values are: 320 x 200 pixels (320 horizontal and 200 vertical pixels) for a PC-style, four-color color graphics adapter (CGA) display; 640 x 200 pixels for a PC-style, two-color CGA display; and 640 x 350 pixels for a PC-style sixteen-color enhanced graphics adapter (EGA) display.

The mechanics of setting the graphics display mode desired and plotting a single point on the display are hardware-dependent, and will be left to the user to determine. Screens 3 and 4 of the accompanying listing contain all the machine-specific primitives for PCs and clones with compatible BIOS ROM chips. They are formatted to use the public-domain 8088 assembler cited[1]. These screens will obviously have to be modified for use on other machines.

```
SCREEN #3
\ "PC" COMPATIBLE EGA, CGA, AND TEXT MODES
HEX   \ Machine specific -- change for your machine!!
CODE SET-CGA-MODE   ( -> )     \ Set mode and clear screen
   AX , # 0004 MOV    10 INT     \ 320 x 200 in 3 colors
   NEXT JMP   END-CODE
CODE SET-CGA-HIRES-MODE   ( -> )     \ Set mode and clear screen
   AX , # 0006 MOV    10 INT     \ 640 x 200 in 2 colors
   NEXT JMP   END-CODE
CODE SET-EGA-MODE   ( -> )     \ Set mode and clear screen
   AX , # 0010 MOV    10 INT     \ 640 x 350 in 16 colors
   NEXT JMP   END-CODE

CODE SET-TEXT-MODE   ( -> )          \ 80 column text
   AX , # 0003 MOV    10 INT
   NEXT JMP   END-CODE
DECIMAL
```

```
SCREEN #4
\ "PC" COMPATIBLE POINT PLOT FOR EGA AND CGA
HEX    \ Machine specific -- change for your machine!!
\ Note that fancier direct screen access assembly language
\   programming can *SIGNIFICANTLY* speed up point plotting
\   at the cost of loss of generality.

CODE PLOT-POINT    ( X Y COLOR -> )  \ Plot a single point
   AX POP    DX POP    CX POP      BX , BX XOR  ( page 0 for EGA )
   AH , # 0C MOV     10 INT
   NEXT JMP   END-CODE

DECIMAL
\ XMAX,YMAX delimit screen boundaries
319 CONSTANT XMAX       \ Change to 639 for EGA or CGA/HIRES
199 CONSTANT YMAX       \ Change to 349 for EGA
  4 CONSTANT #COLORS    \ Change to 16 for EGA , 2 for CGA/HIRES
```

```
SCREEN #5
\ VARIABLE DECLARATIONS, MOVE-CURSOR, SPECIAL BRESENHAM POINT
DECIMAL
VARIABLE XNOW     \ (XNOW,YNOW) is current cursor location
VARIABLE YNOW     \            (0,0) is top left corner of CRT
VARIABLE COLOR    \ current line draw color
  1 COLOR !
\ Variables per Foley & Van Dam, Fund. of ICAD, 1st ed. p 435.
VARIABLE INCR1                VARIABLE INCR2
VARIABLE DX                   VARIABLE DY

: MOVE-CURSOR   ( X Y -> )    \ Move cursor location before a draw
   YNOW !  XNOW ! ;
: POINT    ( X Y -> )         \ Point plot using COLOR variable
   COLOR @  PLOT-POINT ;
: B-POINT    ( X Y DELTA -> )  \ For Bresenham line drawing use
   >R  DDUP  POINT  R> ;
```

```
SCREEN #6
\ BRESENHAM LINE DRAW   PRIMITIVES   +X +Y -X -Y
DECIMAL
: +X   ( X1 Y1 DELTA -> X2 Y2 DELTA )
   ROT 1+    ROT ROT ;

: -X   ( X1 Y1 DELTA -> X2 Y2 DELTA )
   ROT 1-    ROT ROT ;

: +Y   ( X1 Y1 DELTA -> X2 Y2 DELTA )
   SWAP 1+    SWAP ;

: -Y   ( X1 Y1 DELTA -> X2 Y2 DELTA )
   SWAP 1-    SWAP ;
```

```
SCREEN #7
\ BRESENHAM LINE FOR   0 < SLOPE < 1
DECIMAL    \ Assume DX and DY are already set up
: LINE0<M<1    ( NEWX NEWY -> )
   DY @  2*  INCR1 !        DY @  DX @ -  2*  INCR2 !
   ( Pick min x )  OVER  XNOW @  >
      IF  ( current cursor at min x ) DDROP XNOW @ YNOW @ THEN
   DDUP POINT
   ( Compute D )  INCR1 @  DX @ -     \ Stack:  ( X Y DELTA ---)
   DX @  0 DO    DUP  0<
      IF    ( D < 0 )      +X      B-POINT    INCR1 @ +
      ELSE  ( D >= 0 )     +X +Y   B-POINT    INCR2 @ +  THEN
   LOOP
   DROP DDROP ;
```

```
SCREEN #8
\ BRESENHAM LINE FOR   1 <= SLOPE < INFINITY
DECIMAL    \ Assume DX and DY are already set up
: LINE1<M<Z    ( NEWX NEWY -> )
   DX @  2*  INCR1 !        DX @  DY @ -  2*  INCR2 !
   ( Pick min y )  DUP   YNOW @  >
      IF  ( current cursor at min y ) DDROP XNOW @ YNOW @ THEN
   DDUP POINT
   ( Compute D )  INCR1 @  DY @ -     \ Stack:  ( X Y DELTA ---)
   DY @  0 DO    DUP  0<
      IF    ( D < 0 )      +Y      B-POINT    INCR1 @ +
      ELSE  ( D >= 0 )     +X +Y   B-POINT    INCR2 @ +  THEN
   LOOP
   DROP DDROP ;
```

```
SCREEN #9
\ BRESENHAM LINE FOR   -1 < SLOPE < 0
DECIMAL    \ Assume DX and DY are already set up
: LINE-1<M<0    ( NEWX NEWY -> )
   DY @  2*  INCR1 !        DY @  DX @ -  2*  INCR2 !
   ( Pick min x )  OVER  XNOW @  >
      IF  ( current cursor at min x ) DDROP XNOW @ YNOW @ THEN
   DDUP POINT
   ( Compute D )  INCR1 @  DX @ -     \ Stack:  ( X Y DELTA ---)
   DX @  0 DO    DUP  0<
      IF    ( D < 0 )      +X      B-POINT    INCR1 @ +
      ELSE  ( D >= 0 )     +X -Y   B-POINT    INCR2 @ +  THEN
   LOOP
   DROP DDROP ;
```

Straightforward Line-Drawing Algorithms

Now that we can assume the availability of a point-plotting word, how can we draw lines? Horizontal and vertical lines are relatively straightforward. For example:

```
: HORIZONTAL-TEST   ( -- )
100 0 DO   I 10 POINT LOOP ;
```

shows that horizontal lines are drawn by merely incrementing an X value for a constant Y value. Similarly, forty-five-degree lines may be drawn by using a word that simultaneously increments both X and Y values, such as:

```
: DIAGONAL-TEST   ( -- )
  100 0 DO
  I I POINT   LOOP ;
```

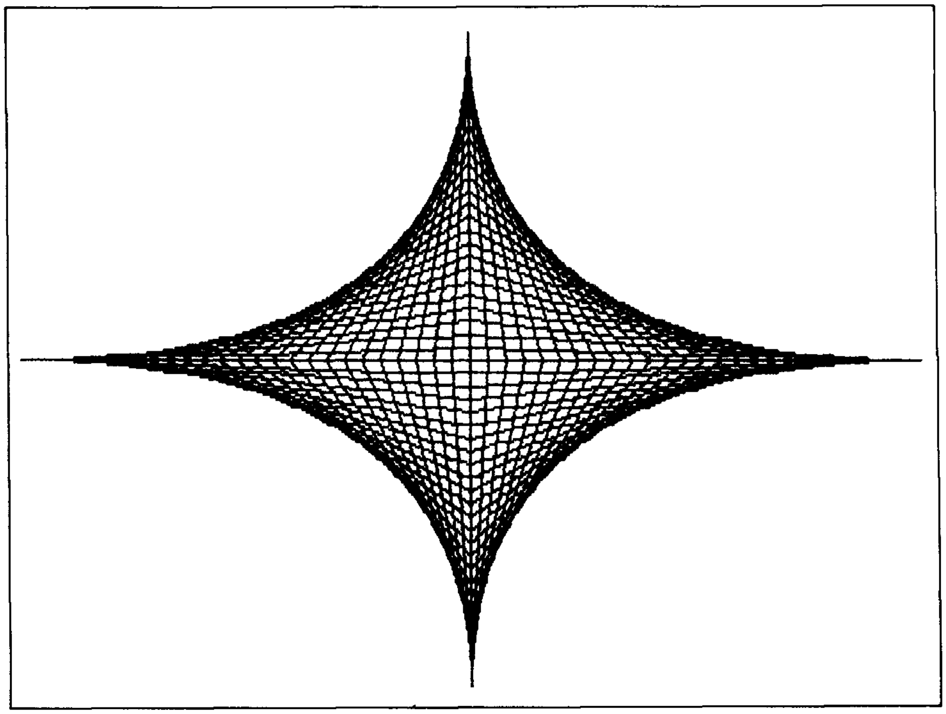
But what about lines that are in-between? A line which spans twice as many X points as Y points would be drawn by:

```
: X=2*Y   ( -- )
  0   100 0 DO
    DUP   I POINT 1+
    DUP   I POINT 1+   LOOP
  DROP ;
```

For a generalized line-drawing word with a slope between zero and one (meaning that the X distance of the line is greater than the Y distance, and that both distances are drawn from smaller to larger numbers), we would have:

```
: GENERAL-LINE  ( X1 Y1 X2 Y2 -- )
  SWAP 4 PICK - SWAP
  3 PICK -    >R >R   100 *    R> R>
  100 3 PICK */    SWAP 1+    0
  DO    3 PICK    3 PICK    100 /    POINT
    SWAP OVER +
    ROT 1+    SWAP ROT    LOOP
  DROP    100 /    POINT ;
```

The above word takes two (X,Y) coordinate pairs as an input, and scales all Y values by 100 to allow for non-integer increments of Y. While this line-drawing algorithm is conceptually straightfor-



Sample GODSEYE output.

ward, it does require a lot of arithmetic. Even if clever scaling factors were chosen to replace most multiplies and divides with shifts and byte-moves, the initial division of the difference between X1 and X2 (sometimes called "delta X" or just plain "DX") by the difference between Y1 and Y2 ("DY") is unavoidable. Another problem is that sixteen-bit scaled integers are not big enough for use on high-resolution screens. In this example, lines that span more than 100 pixels horizontally are improperly drawn.

The Bresenham Algorithm

The Bresenham line-drawing algorithm[2] requires only sixteen-bit integers with addition, subtraction and multiplication by two (shift left) to draw lines. Instead of a scaled, non-integer Y value, the algorithm shown on screen 7 uses the error accumulation term **DELTA** and integer X and Y values. For lines with a slope between zero and one, the algorithm increments the X value for each point, and increments the Y value only if the **DELTA** value is negative. If **DELTA** is negative, a positive value of DY is added to form the new **DELTA** value. If **DELTA** is positive, a negative value based on both DX and DY is used to form a new **DELTA** value.

Of course, slight variations of this algorithm are needed to account for lines with slopes that are not between zero and one. Screens 5 through 13 contain a complete Bresenham line-drawing vocabulary for all line slopes. Horizontal and vertical lines are treated as special cases for greater speed and simplicity.

The vocabulary for using this drawing package is:

SET-CGA-MODE (--)

Places the display in graphics mode. This word may be redefined or renamed as appropriate for your computer.

SET-TEXT-MODE (--)

Returns the display to an eighty-column text mode. This word may be redefined or renamed as appropriate for your computer.

PLOT-POINT (X Y color --)

Plots a single point on the graphics screen. This word may be redefined as appropriate for your computer.

```
SCREEN #10
\ BRESENHAM LINE FOR   -INFINITY < SLOPE < -1
DECIMAL    \ Assume DX and DY are already set up
: LINE-Z<M<-1    ( NEWX NEWY -> )
   DX @  2*  INCR1 !          DX @  DY @ -   2*  INCR2 !
   ( Pick min y )  DUP   YNOW @  >
      IF  ( current cursor at min y ) DDROP XNOW @ YNOW @ THEN
   DDUP POINT
   ( Compute D )  INCR1 @  DY @ -     \ Stack:  ( X Y DELTA ---)
   DY @  0 DO    DUP  0<
      IF    ( D < 0 )       +Y      B-POINT    INCR1 @ +
      ELSE  ( D >= 0 )      -X +Y   B-POINT    INCR2 @ +  THEN
   LOOP
   DROP DDROP ;

SCREEN #11
\ LINE FOR   SLOPE = INFINITY   ( Vertical )
DECIMAL    \ Assume DX and DY are already set up
: LINEZ    ( NEWX NEWY -> )
   ( Pick min y )   DUP  YNOW @  >
      IF  ( current cursor at min y ) DDROP XNOW @ YNOW @ THEN
   DDUP POINT    0  ( dummy DELTA value )
   DY @  0 DO    +Y   B-POINT   LOOP
   DROP DDROP ;

SCREEN #12
\ LINE FOR   SLOPE = 0     ( Horizontal )
DECIMAL    \ Assume DX and DY are already set up
: LINE0    ( NEWX NEWY -> )
   ( Pick min x )   OVER  XNOW @  >
      IF  ( current cursor at min x ) DDROP XNOW @ YNOW @ THEN
   DDUP POINT    0  ( dummy DELTA value )
   DX @  0 DO    +X   B-POINT   LOOP
   DROP DDROP ;

SCREEN #13
\ BRESENHAM PROLOGUE & CALLING ROUTINE
DECIMAL
: LINE  ( XNEW YNEW -> )
   DDUP    ( Extra copy used for final MOVE-CURSOR )
   OVER  XNOW @ -  DUP ABS  DX !   OVER  YNOW @ -  DUP ABS  DY !
   XOR  0<  ( Determine if signs are different )
   DY @  IF       DX @  IF     ( Not horizontal or vertical )
      IF      ( Negative slope )
         DX @  DY @  >  IF  LINE-1<M<0  ELSE  LINE-Z<M<-1 THEN
      ELSE    ( Positive slope )
         DX @  DY @  >  IF  LINE0<M<1   ELSE  LINE1<M<Z   THEN
      THEN
      ELSE  ( Vertical ) DROP  LINEZ  THEN
   ELSE  ( Horizontal )  DROP  LINE0  THEN
   MOVE-CURSOR ;
```

POINT (X Y --)

Same as **POINT**, but without a color value for consistency with **LINE**.

MOVE-CURSOR (X Y --)

Move the current drawing cursor location to the point (X,Y). This word is not called **MOVE** because of possible naming conflicts in some Forth dialects.

LINE (X Y --)

Draw a line from the last cursor position (set by either a **MOVE-CURSOR** or a **LINE** word) to the point (X,Y). The color of the line is determined by the value of the variable **COLOR**.

The demonstration program **GODSEYE** not only draws a pretty picture, but is a good test for the line-drawing algorithm, since it uses lines from each of the different slope-range cases of the line-drawing program.

```
SCREEN #14
\ BRESENHAM  LINE DRAWING TEST PICTURE  -- GODSEYE
DECIMAL
: GODSEYE
  SET-CGA-MODE        \ Change to SET-EGA-MODE for the EGA, etc.
  4  0 DO    3 I -  COLOR !   ( Use this line for CGA )
\ 1  0 DO       1  COLOR !   ( Use this line for CGA/HIRES )
\ 16 0 DO   15 I -  COLOR !   ( Use this line for EGA )
   76 0 DO      75 I -
     150  OVER  2* -    100  MOVE-CURSOR
     150  OVER  25 +   LINE
     150  OVER 2* +  100 LINE
     150  I  100 +  LINE
     150  OVER  2* -    100  LINE
  DROP   3 +LOOP
  ?TERMINAL  ABORT" BREAK IN GODSEYE"
  LOOP     SET-TEXT-MODE  ;
```

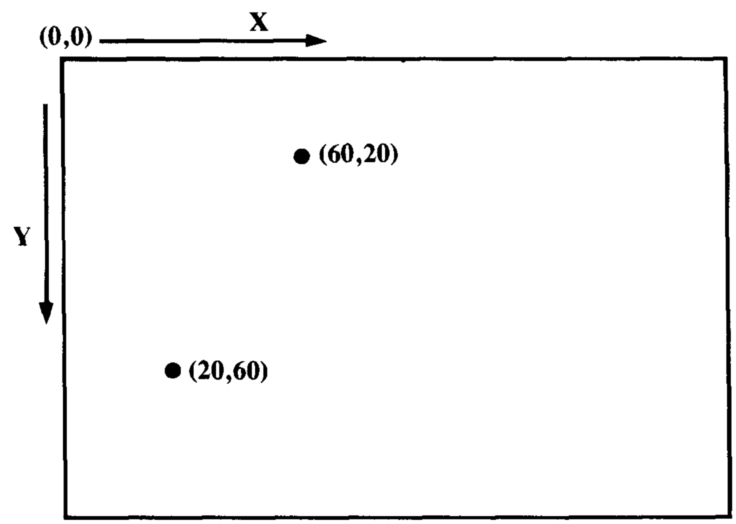


Figure One. Pixel layout on a graphics screen with example points.

Conclusion

The Bresenham line-drawing algorithm is an efficient way to draw straight lines. The lines can be drawn even faster than with the example programs by using techniques such as direct screen-memory access instead of BIOS ROM function calls, and by writing optimized assembly language programs that keep variables in registers instead of in memory. For more information on computer graphics (including mathematical derivations of the Bresenham algorithm), please see the recommended reading list.

In the next issue of *Forth Dimensions*, I will show you how to use these line-drawing words to draw fractal-based landscapes.

Recommended Reading

Fundamentals of Interactive Computer Graphics, J.D. Foley and A. Van Dam, Addison-Wesley, Reading MA, 1982.

Principles of Interactive Computer Graphics, W.M. Newman and R.F. Sproull, McGraw-Hill, New York, 1979.

References

1. *MVP-FORTH Integer and Floating-Point Math*, P. Koopman, Mountain View Press, 1985.
2. "Algorithm for Computer Control of a Digital Plotter," J.E. Bresenham, *IBM Systems Journal*, Vol. 4, No. 1, pp. 25-30, 1965.

Unsigned Division Code Routines

Robert L. Smith
Palo Alto, California

Occasionally, you may find it necessary to write division routines in code. This may occur because you are using a wider precision than is supplied by the machine hardware or your Forth vendor, or perhaps because you are writing your own Forth system. Division routines seem to have a number of pitfalls, and the difficulties of writing these routines are compounded by poor documentation of carry-bit behavior as specified in literature from various manufacturers.

I consider that the most fundamental divide routine is that of the unsigned variety with the numerator having twice the precision of the denominator, and in which the result yields both the quotient and the remainder. In the Forth-83 Standard, it would be called **UM/MOD**. Step-by-step division is frequently accomplished by a series of trial subtractions and shifting. The new bit shifted in at the least-significant-bit position may be the carry bit or its complement, depending on the particular machine. Since the carry (or borrow) bit is frequently poorly documented for the case of subtraction, but almost all machines handle the carry bit the same way for addition, I often find it easier to just complement the divisor before I begin the rest of the routine, and just do trial additions instead of subtractions. If you use subtract or compare operations, but find that the carry bit is the opposite of that desired, you may be able to use the carry bit produced by the computer and merely complement the quotient at the end.

Another pitfall of division routines is the proper handling of the most significant bit of the numerator. Most division routines begin by shifting the double-precision numerator left by one bit before doing the first subtraction. That technique is all right, provided that the most significant bit is not lost in the process. To see the problem, consider a very simple numerator with the most significant bit set to one and all the remaining bits cleared to zero. Take a denominator which is just slightly larger than the high-order part of the denominator. Assume, for example, that we are using the base **HEX**, and that we have a single precision of sixteen bits. The test case would be

```
0  8000  8001  UM/MOD
```

The result of this test should be a remainder of two and a quotient of FFFE on the top of the stack. If the initial divisor, 80000000, were to be shifted left without consideration of the high-order bit, the result would be zero!

Most division routines need a counter to determine when to stop the process. The count-down test may be performed at the end of the shifting process, or in the middle of the shifting process. The latter case is preferable if the count down and testing can be done without disturbing the carry bit. In the former case, it is necessary to stop the main loop one count short and then do a final trial subtraction (or addition), followed by a shift of the carry bit into the quotient only.

Before giving a detailed method for division, it should be noted that there are a wide variety of techniques. If you like this sort of thing, there is a rather fascinating method of non-restoring division in which you either subtract or add, depending on the result of the previous step. Another point worth noting is that if you are implementing a fairly wide multiple precision using the trial subtraction method, the overall speed may be improved by starting your trial subtractions at the most significant byte or word to determine whether or not you really want to do the subtraction.

The preliminary part of the divide routine consists of setting a counter to the number of bits of precision and negating the denominator, usually by popping it off the stack, and placing it in a special register or variable, which I will call **-DEN**. Let us call the next item on the stack **HI** and below that the lower-order part of the numerator, which we can call **LO**. These two items may be popped off the stack, if necessary. The remainder will be developed from **HI**, and the quotient will be developed from **LO**. As a side comment, we note that the natural result of this process is to put the quotient second on the stack, with the remainder on the top. The majority of Forth systems require the reverse of this, necessitating a final **SWAP** to complete the process. One could use an argument based on factoring to suggest that the proper primitive for division would leave the remainder on the top of the stack.

Step 1: Shift **HI** and **LO** left by one bit, with both carry-in and carry-out. If carry-out is zero, go to Step 2. Otherwise, add **-DEN** to **HI** and place the sum in **HI**. Decrement the counter. If the result is non-zero, set the carry bit to one and repeat this step. Otherwise, clear the carry bit and go to Step 3.

Step 2: Make a trial addition of **-DEN** to **HI**. If the carry-out is one, put the sum back into **HI**. In either case, remember the carry bit. Decrement the counter. If the result is non-zero, then go to Step 1. Otherwise, go to Step 3.

Step 3: Shift **LO** only left with carry. If all operands were popped from the stack, first push **HI** on the stack, and then **LO**. Otherwise, do a **SWAP** operation.

That is all there is to it for this routine. Of course, the optimal method will vary from machine to machine. You may be able to obtain a very slight improvement by doing true subtraction on your machine and not doing the initial negation of the denominator.

In my opinion, the arithmetic routines in Forth need to be augmented by division functions which allow double-length divisors. In a version of Forth that I am currently working on, I have added such a routine. It is written in normal 8086/8088 assembly language, and is presented in Figure One as a real example of a division routine. The details of the macro name **HEADER** are unimportant for the purposes of this example.

(See code on page 25.)

DOS File Disk I/O

Charles G. Wilcox
Palo Alto, California

This short article describes a simple interface to PC-DOS used on the IBM PC or equivalent machines. The code is written around MVP-FORTH, but it could be incorporated in virtually any Forth, if desired. A short test indicated that it is compatible with files used by F83.

The words used by the operator are described below. The supporting words are described later.

OPEN Used in the form "**OPEN** filename" to open an existing DOS file or to create a new one. If a new one is being created, then an appropriate message is displayed. An error is returned if the file cannot be opened. With this simple system, only one file can be open at a time.

CLOSE Used in the form "**CLOSE**" to close the presently open file. If the file has been written to, this word updates the directory accordingly (usually only the time has changed). If the file cannot be closed, then an error message is displayed.

DEL Used in the form "**DEL** filename" to delete the file on the default drive, if it can be found. If not, an error message is displayed.

MVP-DISK This patches the execution variable **'R/W** to point to the original **(R/W)** that comes with MVP-FORTH (basic sector I/O).

DOS-DISK This patches the execution variable **'R/W** to point to the new word **DOSR/W**. With these two words, you can toggle back and forth between the two types of disk I/O.

A: This switches the default drive to drive A.

B: This switches the default drive to drive B.

Supporting Words

GFNAME This word parses a filename out of the input stream and moves it to any address you like. Normally, I use **FCB** but in the case of the word **DEL** I use **PAD**. This prevents the file control block from being messed up when I am merely deleting another file.

FCB This is a variable with 2EH more bytes allotted. The file information, including the filename, is stored here.

DOSR/W The code field of this word is put into the execution variable **'R/W** to replace the usual word **(R/W)** used in MVP-FORTH. This word uses the same input as any other R/W word, namely address, block number and either a true flag (indicating a read is desired) or a false flag (indicating a write function is needed).

How It Works

With PC DOS, there are two ways of interfacing to disk files. One, the older method, uses a file control block (FCB). The newer method uses device handles. The newer method allows one to direct I/O easily. I chose the older method, since it was conceptually simpler for me. In this method, one creates a file control block and uses the address of this block as the entry address to a series of interrupt 21H calls. The register AH is set to the code for the command and then interrupt 21H is performed. The error code is usually returned in AL. MVP-FORTH uses the word **SYSCALL** to accomplish this. I use eight such interrupt calls as follows:

open file	0FH
create file	16H
close file	10H
delete file	13H
set disk transfer address	1AH
set default drive	0EH
read random record	21H
write random record	22H

The code should be self explanatory, and is shown in the accompanying screens.

One word of caution, since the parsing word is quite simple: one must use a period after the filename. I also use an extension. One could enhance this operation if one wanted to spend the time at it. If a file has no extension, don't worry; I fill the FCB with blanks, which are okay. Anyway, I usually type something like "OPEN FORTHXX.SCR".

In the newest versions of Forth, the block sizes are one kbyte. I therefore set the record size to one kbyte when I open the file. This is done by storing 400H in the file control block starting at offset 0EH.

Since the word **R/W** in Forth requires (address b# f) for input parameters, I use the address for setting the transfer address. The block number is then put in the relative record number position in the file control block (offset 21H) and then either random read or random write is called, depending on the flag. Any disk I/O error is trapped and a message is displayed.

Use this system as follows:

OPEN FORTHXX.SCR
(opens a new file FORTHXX.SCR)

99 BUFFER DROP UPDATE FLUSH
(write dummy data to disk)

Now you have created a DOS file of 100 kbytes size. Use the familiar words **BLOCK, BUFFER, LIST, LOAD, INDEX,** etc., and they will work as usual.

I am not including any code that will allow you to transfer data from a non-DOS file to a DOS file, but it is very easy to create by using the words **MVP-DISK** and **DOS-DISK** to switch types and then **BLOCK**, **UPDATE** and **FLUSH**. You can figure that out for yourself.

Possible Enhancements

By reading through the DOS manual, one could figure out how to use the file handle system, which is simpler to use once you learn it. It is possible, using the newer file system (indeed, it can be done with the old one, if more than one file control block is available), to have more than one file open at a time. Then data from one file could be transferred conveniently to another file. But in the interest of simplicity, what is presented is adequate for a lot of programming needs. Another enhancement would be to display the DOS directory.

Conclusion

This was very simple code to get working, taking about two hours to accomplish. It allows you to access DOS files for whatever reason you choose. DOS files not created by Forth are easily inspected. Just remember that block zero of any file contains the first one kbyte of the file. When reading a file less than one kbyte in length, you will get a disk read error. Don't worry, though: block zero will still contain the data, and the remainder of the block will be filled with zeroes.

This also answers the question a lot of non-Forth programmers raise: "Why can't Forth talk to DOS files?" My answer to them is, "It can, and here is the code." Then I ask them about trying to convert DOS-created files to non-DOS format. That usually stumps them.

```
SCR # 39
HEX \ NEW DOS FUNCTIONS FOR DISK I/0                          20APR86CGW

VARIABLE FCB 2E ALLOT    \ HOLDS FILE CONTROL BLOCK

: GFNAME    ( ADDR --- ) \ GETS THE FILE NAME FROM INPUT STREAM
    DUP >R 30 ERASE      \ CLEARS THE WHOLE FCB TO ZERO
    R@  1+ 0B BLANK      \ SETS THE NAME FIELD TO BLANKS
    2E WORD COUNT 8 MIN R@  1+ SWAP CMOVE     \ GETS THE FILENAME
    BL WORD COUNT 3 MIN R>  9 + SWAP CMOVE ; \    "     "   EXT

\ MUST USE A PERIOD AFTER FILENAME BUT NO EXT IS REQUIRED

: DEL        ( --- )       \ " DEL filename "
    PAD GFNAME
    13 PAD SYSCALL 0FF AND
    ABORT" file not found" ;
```

```
SCR # 53
HEX \ NEW DOS I/O                                             20APR86CGW

: OPEN       ( --- ) \ " OPEN filename "
    FCB GFNAME
    0F FCB SYSCALL 0FF AND       \ TRIES TO OPEN THE FILE
    IF  ." new file "
        16 FCB SYSCALL 0FF AND  \ CREATES NEW FILE IF NEEDED
        ABORT" error opening file"
    THEN
    400 FCB 0E + ! ;             \ SETS THE RECORD SIZE TO 1K

: CLOSE      ( --- ) \ dont need filename here
    10 FCB SYSCALL 0FF AND
    ABORT" error closing file" ;
```

```
SCR # 54
HEX \ NEW DOS I/O                                             18APR86CGW

: A:
   0E 0 SYSCALL DROP ;      \ SELECT DRIVE A FOR DEFAULT
: B:
   0E 1 SYSCALL DROP ;      \ SELECT DRIVE B FOR DEFAULT

: DOSR/W     ( ADDR B# f --- )    \ REPLACES (R/W) IN MVP
   SWAP FCB 21 + !                \ SETS RAMDOM RECORD NUMBER
   1A ROT SYSCALL DROP            \ SETS TRANSFER ADDRESS
   IF 21 FCB SYSCALL 0FF AND      \ RAMDOM READ DISK RECORD
      ABORT" disk read error"
   ELSE 22 FCB SYSCALL 0FF AND \ RANDOM WRITE DISK RECORD
      ABORT" disk write error"
   THEN ;
```

(Screens continue on page 25.)

FORTH INTEREST GROUP MAIL ORDER FORM

P.O. Box 8231 San Jose, CA 95155 (408) 277-0668

MEMBERSHIP
IN THE FORTH INTEREST GROUP

109 - MEMBERSHIP in the FORTH INTEREST GROUP & Volume 9 of FORTH DIMENSIONS. No sales tax or handling fee. See the back page of this order form.

The Forth Interest Group is a worldwide non-profit member-supported organization with over 4,000 members and 90 chapters. FIG membership includes a subscription to the bi-monthly publication, FORTH Dimensions. FIG also offers its members group health and life insurance, an on-line data base, a large selection of Forth literature, and many other services. Cost is is $30.00 per year for USA, Canada & Mexico; all other countries, $42.00 per year.

The annual membership dues are based on the membership year, which runs from May 1 to April 30.

When you join, you will receive issues that have already been circulated for the current volume of Forth Dimensions, and subsequent issues will be mailed to you as they are published. You will also receive a membership card and number.

HOW TO USE THIS FORM

1. Each item you wish to order lists three different Price categories:

   Column 1 - USA, Canada, Mexico
   Column 2 - Foreign Surface Mail
   Column 3 - Foreign Air Mail

2. Select the item and note your price in the space provided.
3. After completing your selections enter your order on the fourth page of this form.
4. Detach the form and return it with your payment to the **Forth Interest Group**.

FORTH DIMENSIONS BACK VOLUMES

The six issues of the volume year (May — April)

101 - Volume 1 FORTH Dimensions (1979/80)$15/16/18 ______
102 - Volume 2 FORTH Dimensions (1980/81)$15/16/18 ______
103 - Volume 3 FORTH Dimensions (1981/82)$15/16/18 ______
104 - Volume 4 FORTH Dimensions (1982/83)$15/16/18 ______
105 - Volume 5 FORTH Dimensions (1983/84)$15/16/18 ______
106 - Volume 6 FORTH Dimensions (1984/85)$15/16/18 ______
107 - Volume 7 FORTH Dimensions (1985/86)$20/21/24 ______
108 - Volume 8 FORTH Dimensions (1986/87)$20/21/24 ______

FORML CONFERENCE PROCEEDINGS

FORML PROCEEDINGS — FORML (the Forth Modification Laboratory) is an informal forum for sharing and discussing new or unproven proposals intended to benefit Forth. Proceedings are a compilation of papers and abstracts presented at the annual conference. FORML is part of the Forth Interest Group.

310 - FORML PROCEEDINGS 1980 .... $30/33/40 ________
Technical papers on the Forth language and extensions.

311 - FORML PROCEEDINGS 1981 .... $45/48/55 ________
Nucleus layer, interactive layer, extensible layer, metacompilation, system development, file systems, other languages, other operating systems, applications and abstracts without papers.

312 - FORML PROCEEDINGS 1982 .... $30/33/40 ________
Forth machine topics, implementation topics, vectored execution, system development, file systems and languages, applications.

313 - FORML PROCEEDINGS 1983 .... $30/33/40 ________
Forth in hardware, Forth implementations, future strategy, programming techniques, arithmetic & floating point, file systems, coding conventions, functional programming applications.

314 - FORML PROCEEDINGS 1984 .... $30/33/40 ________
Expert systems in Forth, using Forth, philosophy, implementing Forth systems, new directions for Forth, interfacing Forth to operating systems, Forth systems techniques, adding local variables to Forth.

315 - FORML PROCEEDINGS 1985 .... $35/38/45 ________
Also includes papers from the 1985 euroFORML Conference. Applications: expert systems, data collection, networks. Languages: LISP, LOGO, Prolog, BNF. Style: coding conventions, phrasing. Software Tools: decompilers, structure charts. Forth internals: Forth computers, floating point, interrupts, mulitasking, error handling.

BOOKS ABOUT FORTH

200 - ALL ABOUT FORTH . . . . $25/26/35 ________
Glen B. Haydon
An annotated glossary for MVP Forth; a 79-Standard Forth.

216 - DESIGNING & PROGRAMMING PERSONAL EXPERT SYSTEMS . . $19/20/29 ________
Carl Townsend & Dennis Feucht
Introductory explanation of AI-Expert System Concepts. Create your own expert system in Forth. Written in 83-Standard.

217 - F83 SOURCE . . . . $25/26/35 ________
Henry Laxen & Michael Perry
A complete listing of F83 including source and shadow screens. Includes introduction on getting started.

218 - FOOTSTEPS IN AN EMPTY VALLEY (NC4000 Single Chip Forth Engine) $25/26/35 ________
Dr. C. H. Ting
A thorough examination and explanation of the NC4000 Forth chip including the complete source to cmForth from Charles Moore.

219 - FORTH: A TEXT AND REFERENCE $22/23/33 ________
Mahlon G. Kelly & Nicholas Spies
A text book approach to Forth with comprehensive references to MMS Forth and the 79 and 83 Forth Standards.

220 - FORTH ENCYCLOPEDIA . . . . $25/26/35 ________
Mitch Derick & Linda Baker
A detailed look at each fig-Forth instruction.

225 - FORTH FUNDAMENTALS, V.1 . . . $16/17/20 ________
Kevin McCabe
A textbook approach to 79-Standard Forth

230 - FORTH FUNDAMENTALS, V.2 . . . $13/14/18 ________
Kevin McCabe
A glossary.

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

233 - FORTH TOOLS . . . . $22/23/32 ________
Gary Feierbach & Paul Thomas
The standard tools required to create and debug Forth-based applications.

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

237 - LEARNING FORTH . . . . $17/18/27 ________
Margaret A. Armstrong
Interactive text, introduction to the basic concepts of Forth. Includes section on how to teach children Forth.

240 - MASTERING FORTH . . . . $18/19/22 ________
Anita Anderson & Martin Tracy
A step-by-step tutorial including each of the commands of the Forth-83 International Standard; with utilities, extensions and numerous examples.

245 - STARTING FORTH (soft cover) . . . $22/23/32 ________
Leo Brodie
A lively and highly readable introduction with exercises.

246 - STARTING FORTH (hard cover) . . $20/21/30 ________
Leo Brodie

255 - THINKING FORTH (soft cover) . . . . $16/17/20 ________
Leo Brodie
The sequel to "Starting Forth". An intermediate text on style and form.

265 - THREADED INTERPRETIVE LANGUAGES . . . . $25/26/35 ________
R. G. Loelinger
Step-by-step development of a non-standard Z-80 Forth.

267 - TOOLBOOK OF FORTH (Dr. Dobb's) . . . . $23/25/35 ________
NEW
Edited by Marlin Ouverson
Expanded and revised versions of the best Forth articles collected in the pages of Dr. Dobb's Journal.

270 - UNDERSTANDING FORTH . . . . $3.50/5/6 ________
Joseph Reymann
A brief introduction to Forth and overview of its structure.

ROCHESTER PROCEEDINGS

The Institute for Applied Forth Research, Inc. is a non-profit organization which supports and promotes the application of Forth. It sponsors the annual Rochester Forth Conference.

321 - ROCHESTER 1981 (Standards Conference) . . . . $25/28/35 ________
79-Standard, implementing Forth, data structures, vocabularies, applications and working group reports.

322 - ROCHESTER 1982 (Data bases & Process Control) . . . $25/28/35 ________
Machine independence, project management, data structures, mathematics and working group reports.

323 - ROCHESTER 1983 (Forth Applications) . . . . $25/28/35 ________
Forth in robotics, graphics, high-speed data acquisition, real-time problems, file management, Forth-like languages, new techniques for implementing Forth and working group reports.

324 - ROCHESTER 1984 (Forth Applications) . . . . $25/28/35 ________
Forth in image analysis, operating systems, Forth chips, functional programming, real-time applications, cross-compilation, multi-tasking, new techniques and working group reports.

325 - ROCHESTER 1985 (Software Management & Engineering) $20/21/30 ______
Improving software productivity, using Forth in a space shuttle experiment, automation of an airport, development of MAGIC/L, and a Forth-based business applications language; includes working group reports.

THE JOURNAL OF FORTH APPLICATION & RESEARCH

A refereed technical journal published by the Institute for Applied Forth Research, Inc.

401 - JOURNAL OF FORTH RESEARCH V.1 Robotics/Data Structures . . . . $30/33/38 ________

403 - JOURNAL OF FORTH RESEARCH V.2 #1 Forth Machines. . . . . $15/16/18 ________

404 - JOURNAL OF FORTH RESEARCH V.2 #2 Real-Time Systems. . . . . $15/16/18 ________

405 - JOURNAL OF FORTH RESEARCH V.2 #3 Enhancing Forth. . . . . $15/16/18 ________

406 - JOURNAL OF FORTH RESEARCH V.2 #4 Extended Addressing. . . . . $15/16/18 ________

407 - JOURNAL OF FORTH RESEARCH V.3 #1 Forth-based laboratory systems and data structures. . . . . $15/16/18 ________

409 - JOURNAL OF FORTH RESEARCH V.3 #3 . . . . $15/16/18 ________

410 - JOURNAL OF FORTH RESEARCH V.3 #4 . . . . $15/16/18 ________

DR. DOBB'S JOURNAL

This magazine produces an annual special Forth issue which includes source-code listing for various Forth applications.

422 - DR. DOBB'S 9/82 . . . . $5/6/7 ______
423 - DR. DOBB'S 9/83 . . . . $5/6/7 ______
424 - DR. DOBB'S 9/84 . . . . $5/6/7 ______
425 - DR. DOBB'S 10/85 . . . . $5/6/7 ______
426 - DR. DOBB'S 7/86 . . . . $5/6/7 ______

HISTORICAL DOCUMENTS

501 - KITT PEAK PRIMER . . . . $25/27/35 ______
One of the first institutional books on Forth. Of historical interest.

502 - Fig-FORTH INSTALLATION MANUAL $15/16/18 ______
Glossary model editor — We recommend you purchase this manual when purchasing the source-code listing.

503 - USING FORTH . . . . $20/21/22 ______
FORTH, Inc.

REFERENCE

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

300 - FORTH 79-STANDARD . . . . . . $15/16/18 ______
The authoritative description of 79-Standard Forth. Of historical interest.

REPRINTS

420 - BYTE REPRINTS . . . . $5/6/7 ______
Eleven Forth articles and letters to the editor that have appeared in *Byte* Magazine.

ASSEMBLY LANGUAGE SOURCE CODE LISTINGS

Assembly Language Source Listings of fig-Forth for Specific CPUs and machines with compiler security and variable length names.

514 - 6502/SEPT 80 . . . . $15/16/18 ______
515 - 6800/MAY 79 . . . . $15/16/18 ______
516 - 6809/JUNE 80 . . . . $15/16/18 ______
517 - 8080/SEPT 79 . . . . $15/16/18 ______
518 - 8086/88/MARCH 81 . . . . $15/16/18 ______
519 - 9900/MARCH 81 . . . . $15/16/18 ______
521 - APPLE II/AUG 81 . . . . $15/16/18 ______
523 - IBM-PC/MARCH 84 . . . . $15/16/18 ______
526 - PDP-11/JAN 80 . . . . $15/16/18 ______
527 - VAX/OCT 82 . . . . $15/16/18 ______
528 - Z80/SEPT 82 . . . . $15/16/18 ______

MISCELLANEOUS

601 - T-SHIRT SIZE ______
Small, Medium, Large and Extra-Large.
White design on a dark blue shirt. . $10/11/12 ______

602 - POSTER (BYTE Cover) . . . . $5/6/7 ______
616 - HANDY REFERENCE CARD . . . . . . FREE ______
683 - FORTH-83 HANDY REFERENCE CARD . . FREE ______

FORTH MODEL LIBRARY

The model applications disks described below are new additions to the Forth Interest Group's library. These disks are the first releases of new professionally developed Forth applications disks. Prepared on 5 1/4" disks, they are IBM MSDOS 2.0 and up compatible. The disks are compatible with Forth-83 systems currently available from several Forth vendors. Macintosh 3 1/2" disks are available for MasterFORTH systems only.

Forth-83 Compatibility IBM MSDOS

Laxen/Perry F83
MasterFORTH 1.0
PolyFORTH® II
LMI PC/FORTH 3.0
TaskFORTH 1.0

Forth-83 Compatibility Macintosh

MasterFORTH

ORDERING INFORMATION

701 - A FORTH LIST HANDLER V.1 . . . . $40/43/45 ______
by Martin J. Tracy
Forth is extended with list primitives to provide a flexible high-speed environment for artificial intelligence. ELISA and Winston & Horn's micro-LISP are included as examples. Documentation is included on the disk.

702 - A FORTH SPREADSHEET V.2 . . . . $40/43/45 ______
by Craig A. Lindley
This model spreadsheet first appeared in Forth Dimensions Volume 7, Issue 1 and 2. These issues contain the documentation for this disk.

703 - AUTOMATIC STRUCTURE CHARTS V.3 $40/43/45 ______
by Kim R. Harris
These tools for the analysis of large Forth programs were first presented at the 1985 FORML conference. Program documentation is contained in the 1985 FORML Proceedings.

704 - A SIMPLE INFERENCE ENGINE V.4 $40/43/45 ______
NEW
by Martin J. Tracy
Based on the Inference Engine in Winston & Horns book of *Lisp*, this volume takes you from pattern variables to a complete unification algorithm. Accompanied throughout with a running commentary on Forth philosophy and style.

706 - THE MATH BOX V.6 . . . . $40/43/45 ______
NEW
by Nathaniel Grossman
A collection of mathematical routines by the foremost author on math in Forth. Extended double precision arithmetic, a complete 32 bit fixed point math package and auto ranging text graphics are included. There are utilities for rapid polynomial evaluation, continued fractions and Monte Carlo factorization.

Please specify disk size when ordering . . . . ______

FORTH INTEREST GROUP

P.O. BOX 8231 SAN JOSE, CALIFORNIA 95155 408/277-0668

Name ____________

Member Number ____________

Company ____________

Address ____________

City ____________

State/Prov. ____________ ZIP ____________

Country ____________

Phone ____________

OFFICE USE ONLY

By ______ Date ______ Type ______

Shipped By ______ Date ______

UPS Wt. ______ Date ______

USPS Wt. ______ Amt. ______

BO Date ______ By ______

Wt. ______ Amt. ______

ITEM #	TITLE	AUTHOR	QTY	UNIT PRICE	TOTAL
109	MEMBERSHIP →				SEE BELOW

SUBTOTAL	
ORDERS OF $50.00 OR MORE RECEIVE A 10% DISCOUNT	
SUBTOTAL	
CA. RESIDENTS SALES TAX	
HANDLING FEE	$2.00
MEMBERSHIP FEE ☐ NEW ☐ RENEWAL $30/42	

☐ Check enclosed (payable to: **FORTH INTEREST GROUP**)

☐ VISA ☐ MASTERCARD

Card # ____________

Expiration Date ____________

Signature ____________

($15.00 minimum on charge orders)

PAYMENT MUST ACCOMPANY ALL ORDERS

MAIL ORDERS
Send to:
Forth Interest Group
P.O. Box 8231
San Jose, CA 95155

PHONE ORDERS
Call 408/277-0668 to place credit card orders or for customer service. Hours: Monday-Friday, 9am-5pm PST.

PRICES
All orders must be prepaid. Prices are subject to change without notice. Credit card orders will be sent and billed at current prices. $15 minimum on charge orders. Checks must be in US$, drawn on a US Bank. A $10 charge will be added for returned checks.

POSTAGE & HANDLING
Prices include shipping. A $2.00 handling fee is required with all orders.

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

SALES TAX
Deliveries to Alameda, Contra Costa, San Mateo, Los Angeles, Santa Cruz and San Francisco Counties, add 6½%. Santa Clara County, add 7%; other California counties, add 6%.

3-1-87

(Continued from page 20.)

```
SCR # 70
 HEX \ DOS I/O CONTINUED                                    20APR86CGW

 : MVP-DISK      \ CHANGES EXECUTION VARIABLE TO <R/W>
     ' <R/W> CFA 'R/W ! ;

 : DOS-DISK      \ CHANGES EXECUTION VARIABLE TO DOSR/W
     ' DOSR/W CFA 'R/W ! ;

 \ Note that the above works for forth 79 standard.
 \ For forth 83 standard change ' to [']  and delete CFA.
 \ Since this code was written primarily for MVP users
 \ this comment is superfluous.




end of file ok
```

(Continued from page 18.)

```
; UDMOD/   Divide quad by double. ( uquad uddiv -- udquot udrem )
          HEADER   UDMOD/
UDDIV:    POP      CX                  ; DenominatorHi
          POP      DX                  ; DenominatorLo
          POP      AX                  ; AccumulatorHi
          POP      BX                  ; AccumulatorLo
          MOV      BP,32               ; Set count to 32
          PUSH     BP                  ; Keep the count on the stack.
          MOV      BP,SP               ; Point to stack
          CLC                          ; Not really needed
UD1:      RCL      WORD PTR [BP+4],1   ; Shift 64 bit Accum left by 1
          RCL      WORD PTR [BP+2],1
          RCL      BX,1
          RCL      AX,1
          JNC      UD2                 ; If no carry, do test subtract.
UD1SUB:   SUB      BX,DX               ; Carry was set: We must subtract.
          SBB      AX,CX               ; AX is the most significant part.
          DEC      BYTE PTR [BP]       ; Decrement the counter
          STC                          ;
          JNZ      UD1                 ; Continue until counter is zero
          JMP      UD3                 ; Go to trailer when nearly done.
;
UD2:      CMP      AX,CX               ; Start comparision at MS word
          JC       UD2CC               ; If carry is set, don't subtract.
          JNZ      UD1SUB              ; If result is non-zero, subtract.
          CMP      BX,DX               ; Otherwise compare LS word
          JNC      UD1SUB              ; If carry is clear, subtract.
UD2CC:    DEC      WORD PTR [BP]       ; Decrement the counter.
          CLC                          ; Clear the carry bit.
          JNZ      UD1                 ; Continue till count is zero.
UD3:      RCL      WORD PTR 4[BP],1    ; Final adjustment of quotient.
          RCL      WORD PTR [BP+2],1
          MOV      [BP],BX             ; Put LS of remainder on stack.
          PUSH     AX                  ; Push MS of remainder on stack.
          NEXT                         ; Normal ending.
;:
```

Figure One. 8086/8088 division routine allowing double-length divisors.

7776 Limericks

Nathaniel Grossman
Los Angeles, California

In 1961, Raymond Queneau — writer, editor, critic, linguistic experimenter, enthusiastic amateur mathematician and logician — published the remarkable and unique *Cent mille milliards de poemes.*[1] Having care for a slight difference in terminology between French and (American) English, the title may be rendered as "One hundred thousand billion poems," that is, 10^{14} poems which are, in fact, sonnets. While the book does not contain the promised 10^{14} sonnets, it does contains ten sonnets (each of fourteen lines) and, by means of an ingenious system of slits in the pages, the reader may progress downward, selecting and displaying lines from any of the ten sonnets. Thus, any one of 10^{14} sonnets can be displayed for reading. Queneau calculated that, alloting forty-five seconds to reading a sonnet and fifteen seconds more to resetting the pages, with eight hours per day of reading for 365 days per year, over 190 million years would be required to read all the possible variants.

Queneau was inspired with the notion of his book while handling a children's book in which leaflets are flipped to depict chimerical animals. But Queneau did not write chimeras. He composed his sonnets so that lines were interchangeable: if two *k*th lines are interchanged, the new sonnets still make sense. All 10^{14} sonnets are readable on their own! (One must form this conclusion inductively. No one, not even the author, could read more than a vanishingly small fraction of the essentially infinite number of possibilities.)

I was lucky to find a copy of this uncommon book in the library of the University of Durham (England) while living for a while in that city. It is worth looking for in your locale, just to see a splendid piece of bookmaking and paper engineering. I was struck with the thought that Queneau's out-of-print work could be made available to all by means of a simple computer program that would print out as many of the sonnets as the reader wished. Unfortunately, my command of literary French is not great, and I would prefer to savor the sonnets in English. Translation of the 140 lines into English would be a formidable task, given that the meter and rhyme would be severely constrained. Copyright considerations would also intrude. And only a writer of Queneau's power could compose fourteen new, blendable sonnets in English to his plan.

Nevertheless, I found a way to realize the project in miniature. While I shy away from sonnets, I do not hesitate at creating limericks, doggerel as they may be. Therefore, I wrote lines sufficient for creating a six-fold family of limericks, $6^5 = 7776$ in all. I had no pretensions toward creating poems of literary merit: that must be left to the likes of Queneau. I was satisfied if every limerick made "reasonable" sense. To ensure that, it was necessary to choose a vague theme for each line in turn. The restriction to six samples for each line is, of course, arbitrary, but the possibility of only two rhyme schemes will eventually halt proliferation.

Rather than write a program to display all 7776 limericks (who would want to read all of them?), I decided to generate limericks at random from the store of lines, the better to happen upon amusing combinations. Aleatorical composition is neither novel or disreputable: Mozart, for one, was fond of the technique and experimented with it. Naturally, I wrote the program in Forth, specifically in the standard dialect Forth-83. Two non-standard words peculiar to the implementation I use — **DARK** and **@TIME** — are glossed on the screens. They are inessential, in any event. There are more robust algorithms for generating random integers, but they are hardly necessary here. The program is an exercise in the manipulation of data strings, using **CMOVE**.

Reference

1. Queneau, Raymond. *Cent mille milliards de poemes*, Editions Gallimard, Paris, 1961.

An afterword: Several months after writing the above paragraphs, I discovered that the *Cent mille milliards de poemes* had been reprinted within the last few years, although as yet I have not seen a copy of the reprint. What I have seen and acquired is a copy of *One Hundred Million Million Poems*, published by Kickshaws in Paris in 1983, which is a translation by John Crombie into English of the *Cent mille....* Crombie rendered Queneau's hexametric alexandrines into the Shakespearian pentametric sonnets familiar to English readers, but he claims to have preserved the essence of the Queneauisms. I'll leave assessments of the translator's success to those fully bilingual, but reading some of the assembled poems in English, with their odd and unexpected jumps in place, tense and subject, makes me feel a little less apologetic about my limericks, and a lot more admiring of Queneau's enormous linguistic talents.

Returning by water from Ghent,
My family was drenched as they went,
While the son of the Czar
Reeking of rose attar,
Held a fight with a bibulous gent.

Returning by water from Ghent,
A lover fell into the Trent,
All the folks near and far
Reeking of rose attar,
Engrossed with amassing argent.

```
SCR#    1
\ Poem -- loader screen                              NG 04/03/86
\ Randomly generate 7776 different limericks
\ 1 LOAD brings program in, FORGET MARKER forgets it

: MARKER   ( null action )   ;   \ FORGET MARKER to shuck

2 12 THRU

\ Opening prompt
DARK ( cls and home )  CR
.(    Executing  .POEM <return>  will compose and print a ) CR
.( limerick chosen at random from 7776 possibilities available )
 CR .( to you from this program. ) CR CR
```

```
SCR#    2
\ Random integer generator                                 NG 03/22/86
\ following Anderson and Tracy's "Mastering Forth"

: FLIP   ( n1 --- n2 )   \ interchanges bytes of n1 to form n2
   SPLIT SWAP COMBINE   ;

VARIABLE SEED   \ store an integer to initialize RAND
\ The word @TIME is contained in MicroMotion Forth 83, IBM ver.
\ The next line is an implementation-dependent self-seeding.
@TIME ( d from DOS ) DROP SEED !  ( new seed for each booting )

: RAND    (   --- new random integer )
    SEED @ 5421 * 1+  DUP SEED !    ;

: RANDOM   ( n --- random integer between 0 and n-1 inclusive )
    RAND FLIP SWAP MOD    ;
```

```
SCR#    3
\ Size and length variables                          NG 04/03/86

VARIABLE POEMS/STOCK   6 POEMS/STOCK !

VARIABLE LINES/POEM    5 LINES/POEM  !

VARIABLE CHARS/LINE   40 CHARS/LINE  !

\ Adjust the above three values as required by Scrs# 8-...


VARIABLE #LINESTORE   ( # of bytes to be alloted to LINESTORE )

POEMS/STOCK @  LINES/POEM @  CHARS/LINE @  * *   #LINESTORE !
```

```
SCR#    4
\ Make an initialize a buffer for all the lines      NG 04/03/86

\ make a byte array long enough to hold all the chars of all the
\ lines of all the poems

CREATE LINESTORE  #LINESTORE @  ALLOT

\ blank it out

LINESTORE  #LINESTORE @  BLANK
```

```
SCR#    5
\ Find start of any line of any poem in buffer       NG 04/03/86

: BYTE-OFF ( n1 n2 addr --- addr' = addr + n1*n2 )
\ Compute starting address of n1st line in n2nd poem,
\ given starting address of the buffer of lines
    >R                         \ park addr
    SWAP
    POEMS/STOCK @  *  +        \ "matrix" offset
    CHARS/LINE @  *            \ times #chars/cell in "matrix"
    R> +   ;                   \ byte offset = addr'
```

```
SCR#    6
\ Move line from master list to buffer, print it    NG 04/03/86

: >LS   ( addr count n1 n2 ---    )
\ take string ( addr count ) from stack and move its contents
\ to the n1st line and n2nd poem position in the LINESTORE
    LINESTORE
    BYTE-OFF
    SWAP
    CMOVE   ;

: .LINE   ( addr count ---    )
    -TRAILING TYPE   ;

: RANDOM_POEM   (   --- n; random integer < POEMS/STOCK )
    POEMS/STOCK @  RANDOM   ;
```

```
SCR#    7
\ Print a poem composed at random                   NG 04/03/86

: .POEM   (   ---   )
   LINES/POEM @  0 DO          \ for each line of the poem
     I RANDOM_POEM LINESTORE   \ select a suitable line, move
     BYTE-OFF  CHARS/LINE @    \ to its start in LINESTORE
     .LINE  CR                 \ print it, linefeed
   LOOP
   CR CR   ;
```

```
SCR#    8
\ First lines                                         NG 04/03/86

  [   ( begin interpreting )

" Returning by water from Ghent,"                   0 0 >LS
" Relaxing beneath a blue tent,"                    0 1 >LS
" Repairing a furniture dent,"                      0 2 >LS
" Deciding on fasting for Lent,"                    0 3 >LS
" Forgetting to fasten the vent,"                   0 4 >LS
" Adopting an absurd accent,"                       0 5 >LS

  ]   ( begin compiling )
```

```
SCR#    9
\ Second lines                                      NG 04/03/86

  [   ( begin interpreting )

" I noticed my clothespin was bent,"          1 0 >LS
" The widow was late with the rent,"          1 1 >LS
" A lover fell into the Trent,"               1 2 >LS
" My family was drenched as they went,"       1 3 >LS
" The tourist felt dusty and spent,"          1 4 >LS
" Disturbed to an unknown extent,"            1 5 >LS

  ]   ( begin compiling )
```

```
SCR#   10
\ Third lines                                         NG 04/03/86

  [   ( begin interpreting )

"  Though the little grey car"                      2 0 >LS
"  With the tears of the char"                      2 1 >LS
"  And the men at the bar"                          2 2 >LS
"  All the folks near and far"                      2 3 >LS
"  But except for catarrh"                          2 4 >LS
"  While the son of the Czar"                       2 5 >LS

  ]   ( begin compiling )
```

```
SCR#   11
\ Fourth lines                                      NG 04/06/86

  [   ( begin interpreting )

"  And a worldly-wise lar,"                   3 0 >LS
"  'Cause I smoke a cigar,"                   3 1 >LS
"  Looking up at a star,"                     3 2 >LS
"  Reeking of rose attar,"                    3 3 >LS
"  To a mournful guitar,"                     3 4 >LS
"  Running out of the jar,"                   3 5 >LS

  ]   ( begin compiling )
```

```
SCR#   12
\ Fifth lines                                         NG 04/03/86

  [   ( begin interpreting )

" Saw the ships sail into the Solent."            4 0 >LS
" Met twenty birds flown in from Kent."           4 1 >LS
" Engrossed with amassing argent."                4 2 >LS
" Beneath the entombing cement."                  4 3 >LS
" By sacred devotions repent."                    4 4 >LS
" Held a fight with a bibulous gent."             4 5 >LS



\ finish in the interpreting state
```

(Letters, continued from page 10.)

would. It would be interesting to know if one exists. I think it would be more likely (although also doubtful) that some implementation would balk on **DOES>** being in a separate word, for there is a question as to whether this would be strictly legal in Forth-83: the usage pattern for **DOES>** in the standard glossary might be taken to imply that **DOES>** should itself appear in the compiler word, although **CREATE** can be replaced by a word containing **CREATE**.

Sincerely yours,

Victor H. Yngve
Chicago, Illinois

Antecedent Sieve

Dear Marlin,

I was happy to see my improved version of the sieve benchmark in *Forth Dimensions* (VIII/4, page seven). I'm confused, however, by the label "Noyes' Sieve."

This implementation grew out of discussions between Dean Sanderson and myself in 1982. Enclosed please find a copy of page nine from *Dr. Dobb's Journal* (September 1983), in which this algorithm first appeared in print.

Learning and Living,

Don Colburn
Rockville, Maryland

Inverse Video and TI-FORTH

Richard Minutillo
Roslindale, Massachusetts

I had always thought of inverse video as something other computers could do but my TI-99/4A could not. When I obtained TI's Microsoft Multiplan and saw that program's use of inverse video, I realized the obvious: while TI's built-in video firmware cannot provide an inverse image, software can be written to do the trick. TI-FORTH makes the trick easy.

One approach to writing an inverse video routine makes use of an internal memory map which leaves all 256 ASCII characters available for definition and display. This is the approach taken by Multiplan, and there is plenty of room in that configuration to define a whole alternative set of inverted characters between characters 128 and 256. Since the VDP processor thinks of the screen image as a long array of single byte values, it is an easy trick to just find the values which define the section of the screen you want to invert, and then add the 128 offset to each byte to obtain the inverted characters.

Unfortunately, there are several drawbacks to that approach. First, you have to store the ninety-five inverted character definitions in your TI-FORTH routine, and that's a lot of wasted space. Second, you have very few characters available for graphics, if you want them. Finally, it is extremely inelegant.

A relatively compact, and much more elegant, solution can be written in TI-FORTH. I found it so valuable that I added it to my personalized "BSAVEd" kernel, so it can be available for all my applications as I write them.

Here's an outline of the task:

1) Define the screen segment to be inverted by position and length.
2) Read the defined segment of the screen image byte array into a character buffer.
3) Read the eight-byte character definitions of each character to be inverted into a pattern buffer.
4) Invert the pattern buffer.
5) Write the inverted patterns consecutively into VDP memory to redefine a

```
SCR #68
( inverse video / FORTH translation from assembly language
  stack:  row col len - |    RGM 090584)   BASE->R DECIMAL

0 VARIABLE CARBUF 39  ALLOT           ( character buffer )
0 VARIABLE INVBUF 319 ALLOT           ( pattern buffer )
0 VARIABLE LOC    0 VARIABLE LEN      ( variables )

: VARS  LEN ! SCRN_WIDTH @ * + LOC ! ; ( sets variables )

: READSCR  LOC @                      ( screen address in vdp )
           CARBUF                     ( buffer address )
           LEN @                      ( length to read )
           VMBR ;                     ( read section to invert )
                                                             -->


SCR #69
( invert -- screen 2 )

: READCHAR  LEN @ 0 DO                ( loop index )
            CARBUF I + C@ 8 * 2048 + ( address in patt table )
            INVBUF I 8 * +            ( offset into char buffer )
            8                         ( bytes to read )
            VMBR  LOOP ;              ( read charpat into buff )

: INVERTBUF  INVBUF 319 + INVBUF DO   ( loop index )
                 I @ MINUS 1- I !
                        2 +LOOP ;     ( invert entire buff )

: PATTTOVDP INVBUF                    ( inversion buffer )
            3072                      ( address of char #128 )
            320                       ( bytes to write )
            VMBW ;                    ( read to patt table ) -->


SCR #70
( invert - screen 3 )

: WRITESC  LEN @ 0 DO
                 I 128 +              ( character to write )
                 LOC @ I +            ( location on screen )
                 VSBW LOOP ;          ( re-write screen )


: INVERT  VARS READSCR READCHAR INVERTBUF PATTTOVDP WRITESC ;

: REVERT  CARBUF LOC @ LEN @ VMBW ;

: INVERTS >R VARS READSCR READCHAR INVERTBUF PATTTOVDP WRITESC
          R> 0 DO REVERT 2000 0 DO LOOP WRITESC 2000 0 DO LOOP
                  LOOP REVERT ;
                                R->BASE
```

predetermined set of unused ASCII characters.
6) Write an appropriately sized section of redefined characters into the screen image array at the correct location.

By use of variables to hold the screen location and the segment length, and a character buffer to hold the original screen segment, reversal of the inversion is simple: rewrite the character buffer to the saved screen location.

The program outlined above is easily implemented in TI-FORTH. The version presented here is actually a direct conversion from TMS 9900 assembly language. The set of "system synonyms" provided in TI-FORTH makes it possible to translate many machine language tasks without using the separate (and somewhat cumbersome) **ASSEMBLY** and **CODE** vocabularies. All kinds of VDP memory manipulation is possible, and since this program is almost entirely VDP manipulation, all you need to do is make sure the **-SYNONYMS** screen is loaded before you load the screens listed here.

The program listing is largely self documenting and straightforward. Enter a row number, a column number and a segment length on the stack followed by the word **INVERT** and you get inverse video. **REVERT** restores the original characters. **INVERTS** uses the same primitives to repeatedly flash the inversion, and requires additionally the number of flashes on the stack. Use of the user variable **SCRN_WIDTH** to convert row and column to screen location means the procedure will work in **GRAPHICS** or **TEXT** modes. It will not work in bit-mapped or multi-color modes, obviously.

Notice that extremely deep stack manipulations are avoided by the simple expedient of using the **LOC** and **LEN** variables, and the word **VARS** to set those variables *once* from the initial stack. The information could be passed on the stack, but that would make for a much less elegant solution, and for a listing far less easy to read. The definitions as presented provide no error checking, and allow for a maximum of forty characters.

Enter inappropriate values on the stack at your own risk! The original machine-language version was meant to link with TI's Extended BASIC and could use only characters 128 through 143. The longer segment lengths in TI-FORTH are achieved at the cost of increased memory size for the buffers. One could easily modify the buffers and maximum segment length to conserve memory.

State of the Standard

Marlin Ouverson
La Honda, California

Forth standards have arisen, throughout the history of the language, from self-governing committees comprised of expert users of Forth. Participation was open, and becoming a voting member was a matter of meeting minimal requirements. Coming from different backgrounds, these experts often had deeply vested opinions about what should and should not be part of a common Forth kernel, about how those functions would operate and what their names would be, and about standardization itself. The Forth Standards Team (FST) has been the arena in which these elements converge and, at times, diverge. The team has had a benign relationship with the Forth Interest Group, but operates independently. And while it has been the subject of harsh criticism, it has also received a great deal of praise.

What have been the rough spots? One easy target is the malleability that permits Forth to become what each programmer or developer needs (or wants) it to be. More generality and less bulk seem to be called for in a Forth standard than in languages frozen at the moment of creation: Forth systems grow with their users, and those users may resent being told that the programs they develop are non-standard. And creating *new* Forth standards brings the added wrath of both vendors and users if it creates incompatibilities with previously standard systems.

Most of the history of Forth standards has been recorded in these pages and elsewhere, in articles and letters to the editor. We will not attempt a historical summation, but present a sampling of the ideas in active circulation at this time. The amount of material precludes reproduction *in toto*; what follows is a general survey, quoting liberally from documents in our files. To get a reading on the opinions of one cross-section of the Forth community, see "FORML '86 in Review," elsewhere in this issue. We must also acknowledge that, for many people working independently or on some in-house systems, the issue of standards may be only of secondary importance. If, however, this topic is of concern to you, the best way to be fully informed and to participate is to make contact with the people and organizations directly involved.

ANSI Standard Requested

Elizabeth Rather, President of FORTH, Inc., wrote in December to say that a project proposal for an ANS Forth had been filed with ANSI. The group that filed the proposal consisted of Ms. Rather (also an FST member); Don Colburn (FST member, Creative Solutions, Inc.); W.B. Dress (Oak Ridge National Laboratory); Ray Duncan (Laboratory Microsystems, Inc.); Burt Feliss (IBM Corporation); Charles Moore (inventor of Forth, Computer Cowboys); Dean Sanderson (FST Referee, FORTH, Inc.); Gerald Shifrin (MCI Telecommunications Corp.); and Martin Tracy (FIG board member, FORTH, Inc.).

Ms. Rather wrote, "To date we have not heard from ANSI. If and when they do form a Technical Committee for Forth, it will be publicly announced according to their standard procedures, and everyone who is interested and willing to make the required commitment will be able to participate.

"According to ANSI rules, a voting member of a technical committee pays a fee of $175 to ANSI and must attend at least two out of three meetings to retain voting status. The first meeting is usually held at ANSI headquarters in Washington, D.C. Subsequent meetings are held in various parts of the country. Meetings typically occur four times a year for four or five days each. The C committee has been working for five years."

First of all, this means that ANSI has to accept the proposal. And the proposal group does not intend revolution, for the formal proposal states, as the first item in the program of work, "Identify and evaluate common existing practices in the area of the Forth programming language." Under the category of implementation impacts, the proposal points out current incompatibilities among popular Forth dialects and says, "While the Forth-83 Standard has stabilized the language to a great extent, it has proven too restrictive and machine-dependent. Assuming the ANS Forth standard confines itself to such changes as are necessary to resolve the problems in Forth-83, the effect on current practice will be modest." It also projects a five-year useful life of such an ANS standard.

It has long been the view of some that an ANS Forth standard would greatly boost the language's acceptance in the corporate and government world. Others argue that a Forth system stands on its own merits, and that going to ANSI would remove the standardization process too far from the Forth community. The project proposal cited above states, "Preserving machine independence and maintaining a close liaison with any other Forth standardization efforts should prevent problems related to restraint of trade and public interest." It concludes, "If any Forth standard committees are formed by the ISO or IEEE, a close liaison should be formed."

IEEE Action Requested

George Shaw of Shaw Laboratories, Ltd., points out that there is more than one route to an ANSI standard. In one letter, he said, "It took the thirty or so individuals directly involved (and probably several times as many lobbyist and mail participants) in Forth-83 to represent the diversity of implementations and usage. Some important considerations may only have been represented by a single individual.... Considering a standards group with such a small number of participants would end up standardizing a particular group of vendor's implementations at the expense of others. The CBEMA effort, I fear, will produce such a small group."

Shaw explained that ANSI itself doesn't create standards, but endorses them. It is primarily concerned with whether a standards document was obtained from one of their usual channels, like CBEMA (the route chosen by

Rather, et al.) and IEEE. The actual procedure for creating a standard document is dictated by the particular channel. Shaw feels that IEEE can provide for participation and meaningful input from a broader cross-section of the Forth community. Under its own rules, participation by mail must be allowed, and individuals participating by mail are allowed to vote. Teleconferencing and other options are available in consideration of the difficulties of individual participation. CBEMA requires four to six meetings per year, of four or five days each; and the voting membership is virtually restricted to organizations and businesses, not individuals. Shaw's letter continues, "Meeting [CBEMA] requirements to maintain voting privileges would cost approximately $3000 a year in dues and travel expenses, not to mention lost wages or use of vacation time. . . ."

Shaw feels that a CBEMA effort to developing an ANS Forth standard is unnecessarily restrictive, considering how widely expertise is distributed throughout the Forth community. He said that to CBEMA, in a letter asking them not to approve the proposal they received. He points out that the group who wrote the proposal is made up mostly, if not entirely, of current or former employees, customers or subcontractors of FORTH, Inc. He wrote, "They may be well intentioned, but I do not believe this group represents the interests of the Forth community and vendors at large."

This matter was presented at a January meeting of the Microcomputer Standards Committee of the IEEE. Shaw says, while the members of that committee "wished to avoid the possibility of and the political problems involved in having a joint IEEE/CBEMA committee . . . the group voted with no dissension (nineteen yea, four abstentions) to untable and replace a motion made in 1981 for a PAR (program action request) and to additionally request that the members who are also voting members in CBEMA vote against approval of the ANS Forth project." In the IEEE, a PAR is the first step in getting a standard project going.

Forth Standards Team

The above actions may have been prompted by dissatisfaction with the Forth-83 Standard itself, with the process used by the Forth Standards Team or with the continuing unrest in some parts of the Forth community over standardization in general. Vocal dissent over the latest standard seems to have found a home on the East Coast Forth Board (703-442-8695, up to 2400 baud). Sysop Gerald Shifrin sent me standards discussions archived on diskettes that, when printed, amounted to a stack of paper larger than most book manuscripts. I sent a copy of the diskettes to Guy Kelly, FST chairman, to get his reactions.

According to Kelly's analysis, much of the debate over Forth standards on the East Coast Forth Board has been from a vocal few (two participants together account for nearly half the 780 messages; the overall average is twenty-eight messages per participant). Most of the messages fall into a few categories, the first of which is complaints about Forth-83. On the technical side, dissension focuses primarily on floored division, **DO LOOP**s, **FIND**, alleged ambiguities and, in particular, new or modified actions assigned to word names already used. About the last item Kelly says, "Giving old names new meanings was considered the most offensive action that was taken. I agree that it was a radical step and one which should never be repeated! However, it was not done in ignorance, but only after a great deal of careful consideration.

"Something that seems to be completely overlooked in the current discussions is that all the attempts to produce a standard prior to Forth-79 were preliminary gropings and that Forth-79 was fatally flawed....

"Now if the major vendors had said no to the 'obnoxious' changes between the 1979 and the 1983 standards, the standards team probably would have produced a somewhat different Forth-83 Standard (we had received twenty-two yes votes and zero no votes from the twenty-six voting members when the standard was finally released)."

Other topics of discussion from the electronic, standards debate include organization of the FST, suggestions for future standardization efforts and specific work needed (such as surpassing sixteen bits, local or stack variables, data and programming structures, bit manipulation, vectored I/O, quans and transient headers). Extensions have been requested to provide floating-point math, operating system interfaces, files, graphics, strings and math/statistics packages.

Kelly observes that most disliked by the electronic conferees are: floored division, dumb tick ('), smart tick, new **LEAVE**, Forth-83, Forth-79, the small wordset and new actions associated with old word names. He contrasts those with the things conferees have said they like: floored division, dumb tick, smart tick, new **LEAVE**, Forth-83, Forth-79 and the small wordset.

Regarding FORTH, Inc.'s part in the history of Forth standardization, Kelly says, "FORTH, Inc. was forcefully involved in the standards efforts. They hosted the 1977 and 1978 meetings and had three or four participants ... at every standards meeting. FORTH, Inc. has tended to track the various standards, and because Forth is still evolving, the standards have also tended to track the work at FORTH, Inc. The following quote from a May 1978 FORTH, Inc. bulletin entitled *'FORTH-77' Implementation on FORTH, Inc. Systems* may be of interest:

"'We feel that the adoption of standards is an extremely important step in the growing acceptance of Forth, so long as these represent a "minimum vocabulary" with options rather than being interpreted in a restrictive sense.'

Kelly continues, "The fact that Mr. Moore does not personally feel bound by a standard and is continually evolving his own version of the language he invented is, I believe, all to the good."

Finally, the FST chairman calls our attention to these words from the forward to the *Forth-83 Standard*:

"Forth's extensibility allows the language to be expanded and adapted to special needs and different hardware systems. A programmer or vendor may choose to strictly adhere with the standard, but the choice to deviate is acknowledged as beneficial and sometimes necessary. If the standard does not explicitly specify a requirement or restriction, a system or application may utilize any choice without sacrificing compliance to the standard provided that the system or application remains transportable and obeys the other requirements of the standard."

FORML '86 in Review

Last November, FORML conference guests were accompanied to California's Monterey peninsula by a rare migration of whales. Brief, shoreline walks between sessions yielded few sightings, but the salt air, tide pools and crystal-blue skies were balm enough for already stimulated minds.

The theme publicized in the call for papers was, "Extending Forth Towards the 87-Standard." While the topic drew much reaction, only four of the published papers come under that category. Most of the standards-related material was aired during a special working group. Other session subjects included Forth internals, methods, processors, applications and artificial intelligence. This meant more than thirty individual presentations, a half-dozen working groups, impromptu lectures and numerous demonstrations throughout the weekend.

Attendees this year voted to grant special recognition to the following presentors: David Harralson for his paper, "Extended Forth Control Structures for the Language Requirements of the 1990s"; Martin Fraeman, John Hayes, Robert Williams and Thomas Zaremba for their papers pertaining to the Johns Hopkins Forth chip; and Wil Baden for papers about "charting, escaping, hacking and leaping Forth." These papers will be published along with the complete proceedings of the conference, available soon from the Forth Interest Group.

What follows is an informal survey of conference events. Two of the sessions are synopsized by their respective session chairmen. Then working groups are described, along with the names of their chairmen, and some impromptu talks are highlighted.

Forth Internals Session

The first session, consisting of seven papers on Forth internals, covered a wide range of topics. Alternatives were explored for implementation of integer division, improved methods for numerical analysis, better performance for virtual buffer management in systems with many buffers and extraction of system-specific words in F83 to a separate vocabulary. Results were presented on use of modular programming techniques in Forth, portability of programs between sixteen-bit and thirty-two-bit environments was discussed and comparisons were made between subroutine threading on traditional architectures and on the Novix NC4000.

"Turtles Explore Floored Division," by Zafar Essak. This paper addresses the implications of rounding in integer division. Four methods were tested in the production of circles within a two-dimensional "turtle" drawing package: division floored to zero, dropping decimals, was found to be the only method which consistently drew complete and well-formed circles. Detailed implementation source code and demonstration examples were provided in the paper.

"Datastructure: Interpolar," by Nathaniel Grossman. Tabulated functions, or even very complicated elementary functions, are sometimes conveniently approximated by polynomials. Faster and more stable evaluation is made possible by rearranging the polynomials into Barycentric form. This paper presents theory, implementation and examples of numerical analysis using this method.

"Hashed, Cached Buffers," by Loring Craymer. Typical Forth buffer-management implementations are linear in both search and replacement order. They may, therefore, be impractical for implementations in which a large number of buffers are allocated. This paper presents a uniformly fast, non-linear, access-and-replacement method for a large number of block buffers. Comprehensive discussion and source implementation are provided.

"Zapping the F83 Dictionary," by C.H. Ting. Most words in the **FORTH** vocabulary of F83 systems serve very specific system functions and are unused by most programmers. Relinking such words to a hidden vocabulary results in a cleaner and leaner **FORTH** vocabulary that is more useful to most programmers and friendlier to new users. Discussion and implementation source are provided.

"Modular Forth — Import, Export and Linking," by Stephen Pelc and Neil Smith. Most other languages realize benefits from isolating code into separately compiled and linked modules. This paper describes a commercially available product, developed by the authors, which implements a module system that permits external references and interactive linking. Advantages found by the authors include greater productivity and tighter source control. These benefits were particularly important in large team projects.

"Portability: Sixteen to Thirty-Two Bits," by Stephen Sjolander and Jon Waterman. This paper discusses the process of converting a well-defined application from a sixteen-bit-specific implementation to a more general implementation that is easily tailored for either sixteen-bit or thirty-two-bit systems. The source code example provided is a simple polyFORTH decompiler. A decompiler was chosen as a worst-case example in order to highlight machine dependencies. The paper addresses the difficulties encountered and demonstrates the methods used to overcome them.

"Subroutine Threading," by Robert Illyes. This paper compares and contrasts subroutine-threaded implementation strategies for a traditional architecture (ISYS FORTH on the Apple II), with implementation strategies for a Forth-specific architecture (cmFORTH by Charles Moore for the Novix NC4000). Issues discussed include: branch termination, macro generation and the problem of portability. *—Don Colburn*

Standards Session

David Harralson presented "Extended Forth Control Structures," in which he details a set of operators for generalizing flow-of-control, an interesting contrast to Wil Baden's papers from the previous session. Stephen Pelc suggested changes in number-input conversion and vocabulary switching based on his experience as a vendor. Joel Petersen gave some thoughts on his experience circumventing the

Forth-79 and Forth-83 Standards to work with the twenty-bit Nicolet 1280. Don Colburn stressed that future standards should be independent of stack width, and recommended that text files and local variables be considered. Guy Kelley reported that the Forth Standards Team was not currently active and had no present plans of becoming active. Martin Tracy summarized the progress of an ANSI Forth standardization effort. *—Martin Tracy*

Working Groups

"Forth Engines": Conversation centered around comparisons of the Novix 4000 and the Johns Hopkins chip. The two groups had encountered many of the same design decisions, but their approaches were often different. The future of Forth engines was discussed at length, with a considerable amount of interest and expressed optimism. It appears a considerable amount of resource consumption is required to provide thirty-two bits (Johns Hopkins' is thirty-two, Novix' NC4000 is sixteen). Some discussion mentioned that Forth in general is lacking a good set of benchmarks, a lack deserving considerable community effort to remedy.

"Prolog in Forth," chaired by Louis Odette. Data-base, pattern-matching and search utilities are the basis of Prolog. The manner of Prolog's computation was discussed, and was contrasted to Lisp. After a general survey of the language's features, the group dissected its strengths and weaknesses. Next, discussion centered on the ways in which hardware can be brought to aid Prolog's performance. The chairman observed that even with its limitations, Prolog's inertia in the user base is likely to help it retain dominance in the field of artificial intelligence.

"Financial Planning," chaired by William Ragsdale, reported by Jack Park. What do you do after you've spent your professional career hacking code for others? Exposing a small body of earned cash to the world and hoping to come back with more was the topic for this group. Their views: first, establish a reasonable target rate of return. Then evaluate your risk tolerance, level of personal effort and involvement, inventiveness, cleverness and degree of self direction. The next stage is deciding what will you deal in: real estate, commodities, stock, etc. Timing and leverage (how much cash is exposed) are determining factors, and the degree of risk depends on the range of variance in the particular market's rates.

There are optimizing formulas for all this and that is what the group focused on. A few tools written in Forth aid in financial analysis. While one prays they bring an edge, they are providing another set of tools to feed one's personal insight. Evaluating different methods was an area of concern, and detailed discussion ensued.

"CBBS: FIG On-Line," chaired by Robert Berkey. FIG has been talking for two or three years about getting on a large network. This year, GEnie approached FIG and specifics have been discussed. Invitations to prospective sysops were issued by FIG, their resumes received and evaluated. GEnie sent a contract recently, details of which are being negotiated. These announcements excited the group, though some details still remain unresolved. The working group addressed the contract in detail, the difficulty of access to GEnie by non-U.S. residents posing an area of major concern. The company evidently is showing good intentions in this regard by working with Canada and England to provide services in those countries, at least. GEnie is the largest network in the world, but consumer service to date has been restricted to the United States. It is anticipated that using this service would greatly enhance FIG members' ability to interact, to discuss Forth with the world, have access to files of code, etc.

"Forth Standards," chaired by David Petty. This working group had more agreement than anyone expected, considering the sometimes volatile topic. One issue was the ANSI standard request filed recently; the second was that of the standard as a communications document. The group's concensus was that we do need a standard, if for no other reason than to communicate among ourselves. There was also

(Continued on page 41.)

Checksum More

Len Zettel
Trenton, Michigan

Checksums are numbers that result from doing some kind of logical or arithmetic operations on a string of numbers. The simplest checksum is exactly that, the running total. Checksums are very handy gadgets. They are usually used as a verification that two sets of numbers are the same — same checksums, same numbers, same order.

Klaxon Suralis and Leo Brodie showed us some words that did checksums on Forth screens, treating the characters as their ASCII-valued numbers[1]. The idea was, if you hand-entered a source screen from *Forth Dimensions*, say, you could be reasonably sure you had entered it correctly when your system gave you back the same checksum value that had been furnished. I find it also comes in handy when I can't remember if my hard copy backup is of the latest version. If the checksum is still the same, it is.

So, since checksums will have their maximum usefulness if everybody in the world uses the same scheme, why not just dig out the back issues and use the ones already published? Mostly because of one small problem, which Leo Brodie himself created. Because we are only interested in the letter-perfect accuracy of the source code, the checksum words treated multiple blanks as one blank, and skipped comments, which at the time their article was written only came in the form of text enclosed in parentheses. Nowadays, we all use the backslash as another comment indicator[2].

Not to worry, Forthwrights, our favorite language can rise to this occasion. The accompanying screens show how. Naturally, once I got my sticky little fingers into the definitions, I couldn't help rearranging things in the name of improvement. The functions of the words **VERIFY** and **VER** are the same, and should produce the same CRC (checksum) values as the originals. If anyone comes across an instance where this is not the case, please let me know and we'll see what we can do. That is, I expect results to be the same

In **VERIFY** we start by putting the current values of **BLK** and **>IN** on the return stack for safekeeping (users of fig-FORTH can get compatibility by typing **: >IN IN ;** and **: WORD WORD HERE ;**). Then we **BEGIN** asking **MORE** to checksum, which we do in **DISPOSE** until there isn't any more, at which point we restore **>IN** and **BLK** and then exit.

MORE gets us more to checksum. We skip any comments and come up with the address of the next blank-delimited, non-comment entry. Then we check whether we have come to the end of the screen. The differences between various flavors of Forth force a rather elaborate scheme to get the right ending for everybody. We identify the end as a word with a count less than two and ASCII value less than thirty-three. Forth-83 and Forth-79 systems return a count of zero at the end of a block and fig-FORTH systems return one, so we have both of those covered. The last character could be either a blank or a null, and they are both less for screens in upper case. Various machines take liberties with their codes for lower-case letters, and this can cause problems. For instance, while I can get the right CRC value for Suralis and Brodie's original screen 129 on my Amiga, I cannot for their screen 130, which is lower-case text. **VERIFY** takes a screen number off the stack and places the checksum for that screen there. **VER** displays the checksum of the last screen listed. Note that the checksum is treated as an unsigned number, as Suralis and Brodie recommend.

```
SCREEN #85
( CHECKSUMS FOR SCREENS - ACCUMULATE DISPOSE MORE VERIFY VER )

: ACCUMULATE ( OLDCRC CHAR --- NEWCRC)
 256 * XOR 8 0 DO DUP 0< IF 16386 XOR DUP + 1+
                            ELSE DUP + THEN LOOP ;
: DISPOSE ( CRC ADDR --- NEWCRC)
  COUNT >R SWAP R> 0
  DO OVER I + C@ ACCUMULATE LOOP SWAP DROP BL ACCUMULATE ;
: MORE ( --- ADDR )
 SKIP.COMMENTS DUP COUNT 2 < SWAP C@ 33 < AND
 IF DROP 0 THEN ;
: VERIFY ( SCREEN --- CHECKSUM) BLK @ >R >IN @ >R BLK ! 0 >IN !
 0 BEGIN MORE ?DUP WHILE DISPOSE REPEAT
   R> >IN ! R> BLK ! ;
: VER SCR @ VERIFY U. ;
```

```
SCREEN #86
\ SKIP.COMMENTS
0 CONSTANT FALSE 1 CONSTANT TRUE
: SKIP.COMMENTS ( --- ADDR)
 BEGIN BL WORD DUP 2+ C@ BL =
        IF DUP 1+ C@ DUP 40 =
           IF 2DROP 41 WORD DROP TRUE
           ELSE 92 = IF [COMPILE] \ DROP TRUE
                     ELSE FALSE
                     THEN
           THEN
        ELSE FALSE
        THEN
 WHILE REPEAT ;
```

than thirty-three, so we have both of those taken care of; so it should do for just about everyone. When **MORE** hits the end, it puts a zero on the stack that will act as a false flag for the **WHILE** in **VERIFY**. (This assumes that zero cannot be a valid block buffer address.)

SKIP.COMMENTS is designed to do exactly what its name implies. It skips down the input stream until it comes to something non-blank that is not a comment. Right now it is set up to handle two situations. In the first, it runs into a left parenthesis (ASCII 40). Then it uses **WORD** to find the corresponding right parenthesis (ASCII 41), and soldiers onward. In the other case, it finds a backslash (ASCII 92) and then executes a backslash and continues. Note that a left parenthesis or backslash indicates a comment only when immediately followed by a blank, so we check for that first. I suppose if the use of braces gets more popular, another change will be in order; but we now have comment handling factored out and should be able to deal with any fiendish future schemes right here in the code. (Note to Commodore users: on your machines (C64 and VIC-20, at least), ASCII 92 is a British pound sign. It makes a good substitute for the backslash.)

DISPOSE disposes of a non-comment word. It does the checksum on it. **ACCUMULATE** does the bit-twiddling, character by character. It is exactly the same as the Suralis and Brodie **ACCUMULATE**, so we can get the same answers if a backslash is not involved. As a last glitch, by the way, **VERIFY** will not work the way it ought to on the screen containing the colon definition of backslash. I'm too tired to find a way around *that* one.

References

1. Suralis, Klaxon and Leo Brodie. "Checksums for Hand-Entered Source Screens," *Forth Dimensions* IV/3, pg. 15.

2. Brodie, Leo. *Thinking Forth*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1984.

(Continued from page 39.)

wide awareness of the fact that many self-contained applications of Forth do not require, in themselves, adherence to a standard. Standards can, however, make hiring programmers and maintaining programs easier in a great many cases.

Establishing a good process for collecting, evaluating, disseminating and collating standard-related proposals before voting for them was a major concern. Use of a CBBS or of GEnie was considered a possible solution, but the Forth Standards Team could also issue a yearly, printed collection of the proposals it has received. Then a long period (perhaps five years) of field testing should ensue — to see what gets used and liked — before incorporating anything into the standard. About two-thirds of the entire attendance was in favor of seeing ANSI sanction the next standards effort, even though it is bound to be a difficult process. It was felt that Forth's survival would be enhanced by such a standard. The fact that any ANSI standard could simply define a minimal, "least common denominator" of current Forth systems seemed cause for some optimism.

Impromptu Talks

Kim Harris reviewed theories and methods for formal code inspections and structured walk-throughs. These are important tools for any project manager, resulting in clearer understanding of a program and in clearly written and more easily maintained code.

Stephen Pelc spoke about treating vocabularies as objects, allowing them to be referenced without affecting **CURRENT** or **CONTEXT**. An audience remark suggested storing a search method in the object description.

Geoffrey Inett of British Telecom believes the Forth systems developed for them are truly state-of-the-art. Their one caveat is that they would like the system to look more "normal." They use a file-based system, but don't like having the file come up in screens. Also, having developed a product, they'd like to sell it, but finding applications already done in Forth was difficult. Forth has the reputation of a hacker's language and needs a professional image. Suggestions included a directory of applications using Forth *[See* FD *VIII/5, page 37. —Ed.]*, a guide to utilities and tools available in Forth, independent verification of Forth standard adherence (for customers who want that assurance) and outreach efforts that go beyond the Forth community.

—Marlin Ouverson

U.S.

• ALABAMA

Huntsville FIG Chapter
Call Tom Konantz
205/881-6483

• ALASKA

Kodiak Area Chapter
Call Horace Simmons
907/486-5049

• ARIZONA

Phoenix Chapter
Call Dennis L. Wilson
602/956-7678

Tucson Chapter
Twice Monthly,
2nd & 4th Sun., 2 p.m.
Flexible Hybrid Systems
2030 E. Broadway #206
Call John C. Mead
602/323-9763

• ARKANSAS

Central Arkansas Chapter
Twice Monthly, 2nd Sat., 2p.m. &
4th Wed., 7 p.m.
Call Gary Smith
501/227-7817

• CALIFORNIA

Los Angeles Chapter
Monthly, 4th Sat., 10 a.m.
Hawthorne Public Library
12700 S. Grevillea Ave.
Call Phillip Wasson
213/649-1428

Monterey/Salinas Chapter
Call Bud Devins
408/633-3253

Orange County Chapter
Monthly, 4th Wed., 7 p.m.
Fullerton Savings
Talbert & Brookhurst

Fountain Valley
Monthly, 1st Wed., 7 p.m.
Mercury Savings
Beach Blvd. & Eddington
Huntington Beach
Call Noshir Jesung
714/842-3032

San Diego Chapter
Weekly, Thurs., 12 noon
Call Guy Kelly
619/268-3100 ext. 4784

Sacramento Chapter
Monthly, 4th Wed., 7 p.m.
1798-59th St., Room A
Call Tom Ghormley
916/444-7775

Bay Area Chapter
Silicon Valley Chapter
Monthly, 4th Sat.
FORML 10 a.m., Fig 1 p.m.
H-P Auditorium
Wolfe Rd. & Pruneridge,
Cupertino
Call John Hall 415/532-1115
or call the FIG Hotline:
408/277-0668

Stockton Chapter
Call Doug Dillon
209/931-2448

• COLORADO

Denver Chapter
Monthly, 1st Mon., 7 p.m.
Cliff King
303/693-3413

• CONNECTICUT

Central Connecticut Chapter
Call Charles Krajewski
203/344-9996

• FLORIDA

Orlando Chapter
Every two weeks, Wed., 8 p.m.
Call Herman B. Gibson
305/855-4790

Southeast Florida Chapter
Monthly, Thurs., p.m.
Coconut Grove area
Call John Forsberg
305/252-0108

Tampa Bay Chapter
Monthly, 1st. Wed., p.m.
Call Terry McNay
813/725-1245

• GEORGIA

Atlanta Chapter
Monthly, 3rd Tues., 6:30 p.m.
Computone Cotilion Road
Call Nick Hennenfent
404/393-3010

• ILLINOIS

Cache Forth Chapter
Call Clyde W. Phillips, Jr.
Oak Park
312/386-3147

Central Illinois Chapter
Urbana
Call Sidney Bowhill
217/333-4150

Fox Valley Chapter
Call Samuel J. Cook
312/879-3242

Rockwell Chicago Chapter
Call Gerard Kusiolek
312/885-8092

• INDIANA

Central Indiana Chapter
Monthly, 3rd Sat., 10 a.m.
Call John Oglesby
317/353-3929

Fort Wayne Chapter
Monthly, 2nd Tues., 7 p.m.
IPFW Campus
Rm. 138, Neff Hall
Call Blair MacDermid
219/749-2042

• IOWA

Iowa City Chapter
Monthly, 4th Tues.
Engineering Bldg., Rm. 2128
University of Iowa
Call Robert Benedict
319/337-7853

Central Iowa FIG Chapter
Call Rodrick A. Eldridge
515/294-5659

Fairfield FIG Chapter
Monthly, 4th day, 8:15 p.m.
Call Gurdy Leete
515/472-7077

• KANSAS

Wichita Chapter (FIGPAC)
Monthly, 3rd Wed., 7 p.m.
Wilbur E. Walker Co.
532 Market
Wichita, KS
Call Arne Flones
316/267-8852

• LOUISIANA

New Orleans Chapter
Call Darryl C. Olivier
504/899-8922

• MASSACHUSETTS

Boston Chapter
Monthly, 1st Wed.
Mitre Corp. Cafeteria
Bedford, MA
Call Bob Demrow
617/688-5661 after 7 p.m.

• MICHIGAN

Detroit/Ann Arbor area
Monthly, 4th Thurs.
Call Tom Chrapkiewicz
313/322-7862 or 313/562-8506

• MINNESOTA

MNFIG Chapter
Even Month, 1st Mon., 7:30 p.m.
Odd Month, 1st Sat., 9:30 a.m.
Vincent Hall Univ. of MN
Minneapolis, MN
Call Fred Olson
612/588-9532

• MISSOURI

Kansas City Chapter
Monthly, 4th Tues., 7 p.m.
Midwest Research Institute
MAG Conference Center
Call Linus Orth
913/236-9189

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

• NEVADA

Southern Nevada Chapter
Call Gerald Hasty
702/452-3368

• NEW HAMPSHIRE

New Hampshire Chapter
Monthly, 1st Mon., 6 p.m.
Armtec Industries
Shepard Dr., Grenier Field
Manchester
Call M. Peschke
603/774-7762

• NEW MEXICO

Albuquerque Chapter
Monthly, 1st Thurs., 7:30 p.m.
Physics & Astronomy Bldg.
Univ. of New Mexico
Jon Bryan
Call 505/298-3292

• NEW YORK

FIG, New York
Monthly, 2nd Wed., 7:45 p.m.
Manhattan
Call Ron Martinez
212-749-9468

Rochester Chapter
Bi-Monthly, 4th Sat., 2 p.m.
Hutchinson Hall
Univ. of Rochester
Call Thea Martin
716/235-0168

Syracuse Chapter
Monthly, 3rd Wed., 7 p.m.
Call Henry J. Fay
315/446-4600

• OHIO

Akron Chapter
Call Thomas Franks
216/336-3167

Athens Chapter
Call Isreal Urieli
614/594-3731

Cleveland Chapter
Call Gary Bergstrom
216/247-2492

Cincinatti Chapter
Call Douglas Bennett
513/831-0142

Dayton Chapter
Twice monthly, 2nd Tues., &
4th Wed., 6:30 p.m.
CFC 11 W. Monument Ave.
Suite 612

Dayton, OH
Call Gary M. Granger
513/849-1483

• **OKLAHOMA**

Central Oklahoma Chapter
Monthly, 3rd Wed., 7:30 p.m.
Health Tech. Bldg., OSU Tech.
Call Larry Somers
2410 N.W. 49th
Oklahoma City, OK 73112

• **OREGON**

Greater Oregon Chapter
Monthly, 2nd Sat., 1 p.m.
Tektronix Industrial Park
Bldg. 50, Beaverton
Call Tom Almy
503/692-2811

• **PENNSYLVANIA**

Philadelphia Chapter
Monthly, 4th Sat., 10 a.m.
Drexel University, Stratton Hall
Call Melanie Hoag or Simon Edkins
215/895-2628

• **TENNESSEE**

East Tennessee Chapter
Monthly, 2nd Tue., 7:30 p.m.
Sci. Appl. Int'l. Corp., 8th Fl.
800 Oak Ridge Turnpike, Oak Ridge
Call Richard Secrist
615/483-7242

• **TEXAS**

Austin Chapter
Contact Matt Lawrence
P.O. Box 180409
Austin, TX 78718

Houston Chapter
Call Dr. Joseph Baldwin
713/749-2120

Periman Basin Chapter
Call Carl Bryson
Odessa
915/337-8994

• **UTAH**

North Orem FIG Chapter
Contact Ron Tanner
748 N. 1340 W.
Orem, UT 84057

• **VERMONT**

Vermont Chapter
Monthly, 3rd Mon., 7:30 p.m.
Vergennes Union High School
Rm. 210, Monkton Rd.
Vergennes, VT
Call Don VanSyckel
802/388-6698

• **VIRGINIA**

First Forth of Hampton Roads
Call William Edmonds
804/898-4099

Potomac Chapter
Monthly, 2nd Tues., 7 p.m.
Lee Center
Lee Highway at Lexington St.
Arlington, VA
Call Joel Shprentz
703/860-9260

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

• **WISCONSIN**

Lake Superior FIG Chapter
Monthly, 2nd Fri., 7:30 p.m.
University of Wisconsin
Superior
Call Allen Anway
715/394-8360

Milwaukee Area Chapter
Call Donald H. Kimes
414/377-0708

MAD Apple Chapter
Contact Bill Horzon
129 S. Yellowstone
Madison, WI 53705

FOREIGN

• **AUSTRALIA**

Melbourne Chapter
Monthly, 1st Fri., 8 p.m.
Contact Lance Collins
65 Martin Road
Glen Iris, Victoria 3146
03/29-2600

Sydney Chapter
Monthly, 2nd Fri., 7 p.m.
John Goodsell Bldg.
Rm. LG19
Univ. of New South Wales
Sydney
Contact Peter Tregeagle
10 Binda Rd., Yowie Bay
02/524-7490

• **BELGIUM**

Belgium Chapter
Monthly, 4th Wed., 20:00h
Contact Luk Van Loock
Lariksdreff 20
2120 Schoten
03/658-6343

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

• **CANADA**

Alberta Chapter
Call Tony Van Muyden
403/962-2203

Nova Scotia Chapter
Contact Howard Harawitz
227 Ridge Valley Rd.
Halifax, Nova Scotia B3P2E5
902/477-3665

Southern Ontario Chapter
Quarterly, 1st Sat., 2 p.m.
General Sciences Bldg., Rm. 312
McMaster University
Contact Dr. N. Solntseff
Unit for Computer Science
McMaster University
Hamilton, Ontario L8S4K1
416/525-9140 ext. 3443

Toronto FIG Chapter
Contact John Clark Smith
P.O. Box 230, Station H
Toronto, ON M4C5J2

• **COLOMBIA**

Colombia Chapter
Contact Luis Javier Parra B.
Aptdo. Aereo 100394
Bogota
214-0345

• **ENGLAND**

Forth Interest Group — U.K.
Monthly, 1st Thurs.,
7p.m., Rm. 408
Polytechnic of South Bank
Borough Rd., London
D.J. Neale
58 Woodland Way
Morden, Surry SM4 4DS

• **FRANCE**

French Language Chapter
Contact Jean-Daniel Dodin
77 Rue du Cagire
31100 Toulouse
(16-61)44.03.06

• **GERMANY**

Hamburg FIG Chapter
Monthly, 4th Sat., 1500h
Contact Horst-Gunter Lynsche
Common Interface Alpha
Schanzenstrasse 27
2000 Hamburg 6

• **HOLLAND**

Holland Chapter
Contact: Adriaan van Roosmalen
Heusden Houtsestraat 134
4817 We Breda
31 76 713104

FIG des Alpes Chapter
Contact: Georges Seibel
19 Rue des Hirondelles
74000Annely
50 57 0280

• **IRELAND**

Irish Chapter
Contact Hugh Doggs
Newton School
Waterford
051/75757 or 051/74124

• **ITALY**

FIG Italia
Contact Marco Tausel
Via Gerolamo Forni 48
20161 Milano
02/645-8688

• **JAPAN**

Japan Chapter
Contact Toshi Inoue
Dept. of Mineral Dev. Eng.
University of Tokyo
7-3-1 Hongo, Bunkyo 113
812-2111 ext. 7073

• **NORWAY**

Bergen Chapter
Kjell Birger Faeraas
Hallskaret 28
Ulset
+47-5-187784

• **REPUBLIC OF CHINA**

R.O.C.
Contact Ching-Tang Tzeng
P.O. Box 28
Lung-Tan, Taiwan 325

• **SWEDEN**

Swedish Chapter
Hans Lindstrom
Gothenburg
+46-31-166794

• **SWITZERLAND**

Swiss Chapter
Contact Max Hugelshofer
ERNI & Co., Elektro-Industrie
Stationsstrasse
8306 Bruttisellen
01/833-3333

SPECIAL GROUPS

Apple Corps Forth Users Chapter
Twice Monthly, 1st &
3rd Tues., 7:30 p.m.
1515 Sloat Boulevard, #2
San Francisco, CA
Call Robert Dudley Ackerman
415/626-6295

Baton Rouge Atari Chapter
Call Chris Zielewski
504/292-1910

FIGGRAPH
Call Howard Pearlmutter
408/425-8700

MMS Forth User Groups
(More than 30 locations.)
For further information call:
617/653-6136