FORTH Dimensions

Volume 8, Number 5

January/February 1987
$4.00

Floating-Point Arithmetic

The Ultimate CASE

Classes in Forth

Tracking the Beast

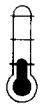
Screenless Forth

National Forth Convention

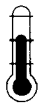
Forth Dimensions
Published by the
Forth Interest Group
Volume VIII, Number 5
January/February 1987
Editor
Marlin Ouverson
Advertising Manager
Kent Safford
Production
Cynthia Lawson Berglund
Typesetting
LARC Computing

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.



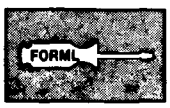
Intermediate; articles and code for more complex applications, and tutorials on generally difficult topics.

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

A Tale of Recursion

Dear Editor,

While reading the very interesting article, "The Point Editor" (VIII/3) by J. Brooks Breeden, I couldn't help noticing the use of **RECURSE** at the end of the word **MENU** in screen seven. This is an example of tail recursion, where the recursive call is made at the end of the function and no more processing comes after the recursive call. Tail recursion can be caught by a smart interpreter, such as LOGO, and turned into iteration for efficient use of the return stack. In LOGO, recursion is the only way to do indefinite loops. I thought, why not have a smart version of **RECURSE** so that I can use tail recursion in Forth without worrying about my return stack overflowing?

See my included screen for a solution. **RECURSE** starts by saving the input stream pointer >**IN** so that it can look ahead. If the next word is ; or **THEN** followed by ;, we have a case of tail recursion and an iterative branch to the beginning of the word being defined is called for. Otherwise, the word's own compilation address is compiled, to allow a recursive call to take place. Finally, the input stream is restored and compilation continues normally. The difference between the two cases is that, at run time, tail recursion avoids using the return stack.

Included are two examples taken from Michael Ham's article, "Recursion" (*Forth Dimensions* VI/4). **GCD** is the greatest common divisor, and is an example of tail recursion. **FACTORIAL** is an example of true recursion, where both stacks pile up and processing occurs both before and after the word **RECURSE**. These definitions seem to work as expected, but if I've overlooked anything, please write and inform me.

Charles Shattuck
Roseville, California

```
Scr#   57.....................................................
  0 \ Efficient tail recursion                    17Oct86 CWS
  1 : RECURSE   >IN @   ' ( next word)  ['] THEN = NOT   \ near end?
  2    IF  DUP >IN !  THEN             \ then restore the input stream
  3    LATEST NAME)       \ compilation address of word being defined
  4    ' ( next word)  ['] : =                  \ at end of definition?
  5    IF  COMPILE BRANCH  >BODY (RESOLVE \ then branch to beginning
  6    ELSE  ,                        \ else call the function recursively
  7    THEN  >IN ! ; IMMEDIATE COMPILE-ONLY   \ restore input stream
  8
  9 : GCD   ( a b -- gcd)   \ an example of tail recursion
 10    ?DUP  IF SWAP OVER MOD  RECURSE  THEN ;
 11
 12 : FACTORIAL   ( n -- n!)   \ an example of true recursion
 13    DUP 1 = NOT  IF  DUP 1- RECURSE *  THEN ;
 14
 15
```

Shattuck Screen

Forth aux Ecoles

Dear Marlin,

I would like to tell you about a French teaching experience in Quebec, Canada. This program at the Collège de Sherbrooke is titled, "Technologie des Systèmes Ordinés." This three-year program aims at forming technicians who can adapt and maintain software, as well as repair microcomputers. The programming is mostly centered on real-time applications, while hardware revolves around chips like pio, sio, pic, crtc, etc. But the students also learn other useful tools like word processing, databases, spreadsheets, communications, CAD and so on. In fact, we try to take the best out of the two worlds of electronics and programming.

Here is how we teach and use Forth. In the first semester of their first year, students follow a basic course on programming logic and the rudiments of Forth, using a network of twenty Compaq Deskpro's. The use of computer graphics is of primary importance, since it motivates the students while permitting them to learn the elementary control structures.

In the second year of this program, students develop real-time applications (in Forth), using concepts such as multi-tasking, an I/O toolbox, code definitions, low- and high-level interrupts, etc. As an example, students last year simulated a railroad crossing control using an STD bus system and I/O modules.

Finally, in their last year, the students have a course on the internals and the extensibility of Forth, including the higher level of metacompilation. In the last semester of that program, students in groups of two have 300 hours to work (with assistance) on a main project. Most of these projects are coming from "real" needs among the region's industries. These projects must be about half hardware and half software to get approval from the instructors. The software must be written in Forth, assembler or both, and is put into EPROMs if necessary. As an example, students last year developed two projects for Ph.D.'s in nuclear physics at the Université de Sherbrooke. One was for data acquisition and control of an electron gun in an experiment about the diffraction of "slow" electrons. Another was the temperature control of a hothouse for growing tomatoes. Of the projects that were eventually put into EPROMs, we can mention a PID temperature control and an ultra-sonic radar with graphic display on a VT-100 terminal.

For our needs, we use a modified version of Laxen & Perry's F83 for the IBM PC (congratulations for your work, guys!). The major changes brought to it were to get rid of the view fields in the structure (because it now loads from normal MS-DOS level 2 text files), the use of binary overlays to speed up the loading of precompiled applications and a complete set of graphics words (including LOGO-like commands).

This year we had a grant from the provincial government and bought a FORCE VME computer equipped with a 68020 (16 MHz) microprocessor. We will drive it with a polyFORTH system. We are expecting a lot of possibilities from this machine. More to come...

But since our actual control projects are mainly done with the Z80, we are using a CP/M network of STD bus stations, duplicating easily the future targets and simplifying the development of stand-alone applications. We have also modified a Nautilus metacompiler to make it F83 compatible and to build EPROM versions of code that was previously tested on the CP/M workstations.

That is what is so fantastic about Forth: you can change it to make it appropriate to your needs!

As an example, lately we wrote two simple words (>**FORTH** and >**ASM**) that permit us to execute high-level words within a code definition. This is very useful within a slow-interrupting system when there are math equations to perform and system status to display on a console (a PID control loop with adjustable parameters, for example).

Other colleges (CEGEP) in Quebec are thinking about switching to Forth, just like we did four years ago. There will probably be a course given by the Université de Sherbrooke in the spring of 1987 for the CEGEP teachers.

We would like very much to exchange information with other institutions about their Forth teaching experiences.

Denis Lambert
Collège de Sherbrooke
Sherbrooke, Quebec
Canada

On-Line Docs for fig-FORTH

Dear Marlin,

Regarding "On-Line Documentation" (*Forth Dimensions* VIII/2), it is a very good idea. I have implemented it. Some fig-FORTH users, however, are going to have some trouble getting it to work. Perhaps I can help.

Mr. Wavrik's definition of **LOCATE** seems to assume that the word **-FIND** leaves a CFA and a flag. The usual fig-FORTH **-FIND** leaves **PFA**, **CNT** and a flag. The count (**NFA**'s count byte contents) is a gremlin floating around in the word **LOCATE**. The word **CFA>SFA** is actually being fed the count, and even after the count is dropped, **CFA>SFA** receives a PFA, not a CFA.

Another difficulty is that the **U<** in the word **LOCATE** should be just plain < in many systems, mine included, as the LFA, NFA, CFA and PFA will all be negative numbers and growing in the right direction to use <.

Also a problem is the assumed decompilation of **CREATE**. Most users will be safe, in that their **CREATE** will be a standard fig-FORTH definition, but mine is not. In those cases where it varies from

```
: CREATE -FIND IF DROP ...;
```

then changing the patch word **XCREATE** to end with the first word in the definition of **CREATE**, instead of with **-FIND**, should work.

The listing shows an application of **LOCATE** that will work. For TI-Forth users, the definition of **SFA-PUT** is:

```
: SFA-PUT BLK , =CELLS HERE ;
```

The code on line one should be compiled until debugging is no longer necessary. If things don't go right, then simply keying in

```
FORGET SFA-PUT (XCREATE)
```

and then reloading will crash the system. Why? Because **CREATE** has already been patched, and compiling the screen again patches it again. Before recompiling, execute **RESTORE**.

Sincerely,

Gene Thomas
Little Rock, Arkansas

```
                    Locate Utility
Listing 1
Screen #16
  0. \ LOCATE, rev. gtAug86 (fig)                        j.j.w. FD 8/2
  1.    ' CREATE @  CONSTANT (C)          : RESTORE  (C) ' CREATE ! ;
  2.
  3. : SFA-PUT   BLK @ ,  -FIND  ;        \ xcreate
  4.    ' SFA-PUT CFA    ' CREATE !       \ patch create with sfa-put
  5.    HERE CONSTANT WALL                \ no sfa's below wall
  6. : SFA ( pfa -- sfa)   LFA 2-  ;      \ lfa,nfa,cfa,pfa dict. order
  7.    \ : SFA  NFA 2-  ;                \ nfa,lfa,cfa,pfa dict. order
  8. : KB? ( blk -- f:blk 0=tf) @ 0=  ;
  9. : SHOW-SCR   @ LIST  ;              \ or:  @ edit
 10. : NIP-CNT  ( cfa cnt f -- cfa f)    SWAP DROP  ;
 11.
 12. : LOCATE  -FIND  NIP-CNT   0= IF  ." Not found"  ELSE
 13.           DUP WALL <          IF ." Not locatable"  DROP ELSE
 14.           SFA DUP KB?         IF ." Block 0"        DROP ELSE
 15.           SHOW-SCR            THEN THEN THEN  ;
```

Thomas Screen

Apologia in Absentia

Dear Marlin,

This letter is intended as an apology to all those who wrote to The Tools Group and never received a reply. The reason for the lack of response was that I never got the letters.

About the time the first ad for The Tools Group came out, I broke up with my girlfriend and sold the house in Desert Hot Springs to her. Although some mail has been forwarded by the post office, I am sure that I did not receive a number of responses. To those writers, I offer my apologies.

The Tools Group was formed to develop and market the Forth we had developed as the tools group for a large Forth project. The most significant feature of our Forth is the large number of extensions (library manager, floating point, etc.).

Looking around at the marketplace, we have decided there are enough versions of Forth in existence. We have decided to adapt our tools to established Forth packages, supplementing the tools those vendors supply. This conversion effort is underway and should be ready for public consumption soon. At that time, we will run our ad in *Forth Dimensions* (with the correct address).

Regards,
Ron Braithwaite
The Tools Group
Forest Falls, California

F83 Compiles Text

Dear Marlin,

In a letter to *Forth Dimensions* (VIII/2), Mr. Ramer W. Streed asked for a program to read and compile F83 code for the IBM PC from a text file. The accompanying screens will do that. The requirements are MS-DOS 2.1 or greater and plain ASCII text files. I hope this is useful for Mr. Streed and your readership.

Sincerely,

Alberto Pasquale
Houston, Texas

Pasquale Screens

```
Scr # 0          B:TEXTLOAD.BLK
  0 \  F83 TextLoad.blk  by Alberto Pasquale              11/15/1986
  1 TextLoad < filename > loads a text file and prints it on the
  2                       screen
  3 Control-table replaces CC (Kernel86.blk scr# 48)
  4 (open-f) and (close-f) requires MS-DOS 2.1
  5 (f-key) reads a byte from an open file into TOS
  6         replaces key to redirect input from the keyboard to a
  7         text file
  8 ?err-0 executes eof if an error is encountered during loading
  9 eof brings the system back to normal and closes the file
 10 control-z indicates that all the file has been loaded and
 11           executes eof
 12 TextLoad opens a file, drops line-feeds 'J
 13          redirects key to make F83 think you are typing the
 14          file at the terminal.
 15


Scr # 1          B:TEXTLOAD.BLK
  0 \  F83 TextLoad.blk  by Alberto Pasquale              11/15/1986
  1 DEFER 'A DEFER 'B DEFER 'C DEFER 'D DEFER 'E DEFER 'F DEFER 'G
  2 DEFER 'I DEFER 'J DEFER 'K DEFER 'L DEFER 'N DEFER 'O DEFER 'Q
  3 DEFER 'R DEFER 'S DEFER 'T DEFER 'V DEFER 'W DEFER 'Y DEFER 'Z
  4 DEFER 'O        ' NOOP IS 'O
  5 ' (CHAR) IS 'A ' (CHAR) IS 'B ' (CHAR) IS 'C ' (CHAR) IS 'D
  6 ' (CHAR) IS 'E ' (CHAR) IS 'F ' (CHAR) IS 'G ' (CHAR) IS 'I
  7 ' (CHAR) IS 'J ' (CHAR) IS 'K ' (CHAR) IS 'L ' (CHAR) IS 'N
  8 ' (CHAR) IS 'O ' (CHAR) IS 'Q ' (CHAR) IS 'R ' (CHAR) IS 'S
  9 ' (CHAR) IS 'T ' (CHAR) IS 'W ' (CHAR) IS 'Y ' (CHAR) IS 'Z
 10
 11 CREATE CONTROL-TABLE   CONTROL-TABLE CC !
 12  ] 0       'A     'B     'C     'D     'E       'F     'G
 13    BS-IN   'I     'J     'K     'L     CR-IN    'N     'O
 14    P-IN    'Q     'R     'S     'T     BACK-UP  'V     'W
 15    BACK-UP 'Y     'Z      CHAR   CHAR   CHAR     CHAR   CHAR  [
```

```
Scr # 2          B:TEXTLOAD.BLK
  0 \  F83 TextLoad.blk  by Alberto Pasquale                11/15/1986
  1 HEX
  2 CODE (OPEN-F)  ( filename-adrr -- handle flag)
  3    DX POP 3D02 # AX MOV 21 INT AX PUSH
  4    U< IF 0 # AX MOV ELSE 1 # AX MOV THEN   1PUSH END-CODE
  5 CODE (CLOSE-F) ( handle -- )
  6     BX POP 3E # AH MOV 21 INT NEXT END-CODE
  7 VARIABLE F-HANDLE  VARIABLE K-BUF
  8 LABEL F-ERROR  0 # AX MOV 1PUSH
  9 CODE (F-KEY)    (  --- n )
 10  F-HANDLE #) BX MOV  1 # CX MOV K-BUF # DX MOV  3F # AH MOV
 11  21 INT F-ERROR JB
 12  CX AX SUB 0<> IF 1A # AL MOV
 13                ELSE K-BUF #) AX MOV THEN
 14  AH AH SUB 1PUSH END-CODE
 15 DECIMAL
```

```
Scr # 3          B:TEXTLOAD.BLK
  0 \  F83 TextLoad.blk  by Alberto Pasquale                11/15/1986
  1
  2
  3 VARIABLE F-NAME  15 ALLOT
  4 : (GET-FNAME)   14 MIN DUP  ROT ROT
  5    F-NAME SWAP MOVE F-NAME + 0 SWAP C! ;
  6 : GET-FNAME BL WORD COUNT (GET-FNAME) ;
  7 : EOF  ['] (KEY) IS KEY ['] (CHAR) IS `J
  8        ['] NOOP IS `O   ['] RES-IN IS `Z
  9        ['] (?ERROR) IS ?ERROR
 10        F-HANDLE @ (CLOSE-F) ;
 11 : CONTROL-Z ." END OF FILE " CR  EOF BACK-UP CR   ;
 12 : ?ERR-0 DUP IF EOF  (?ERROR)  ELSE DROP 2DROP THEN :
 13
 14
 15
```

```
Scr # 4          B:TEXTLOAD.BLK
  0 \  F83 TextLoad.blk  by Alberto Pasquale                11/15/1986
  1
  2 : TEXTLOAD
  3   GET-FNAME F-NAME (OPEN-F)
  4   IF F-HANDLE !
  5        ['] DROP IS `J
  6        ['] (F-KEY) IS KEY
  7        ['] EOF IS `O
  8        ['] CONTROL-Z IS `Z
  9        ['] ?ERR-0 IS ?ERROR
 10   ELSE TRUE ABORT" FILE NOT FOUND" THEN ;
 11
 12
 13
 14
 15
```

A Sense of Place

Last November was one of the busiest months in our history. A tour to exchange technical papers in China, a national Forth convention and a FORML conference all occurred during production of this issue. We try to keep you informed, but details of these events would fill at least two entire issues. Look for convention coverage herein; a brief review of FORML will appear in the following issue, but the entire proceedings will be published separately, as usual, to keep you abreast of useful, new findings and techniques.

At several Forth conferences, I've met representatives from Bell Canada, Stanford University, Johns Hopkins University, British Telecom and Eastman Kodak, to name only a few large sites where Forth is used. Some of those who cannot attend these events personally may still feel that Forth has yet to come into its own in terms of public recognition. They may have outdated notions of Forth's place in the world.

The question, "Why isn't Forth recognized more widely?" has been with us too long. Certainly we cannot hope for from others what we do not grant ourselves. Some very large names indeed have designated Forth as their language of choice for major projects, investing money and manpower in its use. And they receive tangible gains in development time and cost, efficiency, maintainence... Well, it will be best if such Forth users make their own statements. *Forth Dimensions* will tell the stories this year of some installations, large and small, using Forth. We think you'll find it interesting and eye-opening.

This is part of a larger information-gathering project. We hope Forth vendors and programmers will help us to compile the first complete document of Forth's use in all manner of systems and products. We first published a questionnaire a year ago (issue VII/5) which brought many fascinating responses, but still reached only the tip of the iceberg. That questionnaire is reprinted in this issue — please use it yourself and see that copies get passed to non-FIG members who have been involved in Forth projects.

On a final note, the new set of *Forth Dimensions* writer's guidelines is now available from FIG. It provides information that new writers, as well as our regulars, should have on hand. Much of the material in it will also help anyone writing about Forth for other publications. The price is right, so if you would like to write an article, tutorial or technical note, please send for a free copy. We will look forward to hearing from you!

—Marlin Ouverson

Johns Hopkins Correction

Dear Editor,

We would like to point out a factual error in Glen Haydon's article, "The Multi-Dimensions of Forth" (VIII/3). The article, in describing several hardware Forth engines, states that we at Johns Hopkins University's Applied Physics Laboratory "...have taken the basic design of the Novix 4000 device and expanded it to a thirty-two bit processor on a chip." It is true that we have designed a single-chip, thirty-two bit Forth processor, but it is in no way related to the Novix processor. Our processor was independently designed based on our experience with a microprogrammed bit-slice Forth engine our group designed for the Hopkins Ultraviolet Telescope, a part of the ASTRO Space Shuttle mission.

The Novix processor and our processor are radically different in both architecture and implementation. The Novix chip achieves high performance by connecting to external memory via three buses, one for fetching instructions and two for accessing the parameter and return stacks. Our processor uses a more conventional single bus, but caches the top sixteen elements of both the parameter and return stack on chip. Our architecture was influenced by RISC research and has only two instruction formats. The Novix design is implemented in a CMOS gate array. We did a full custom implementation of our design in four-micron SOS CMOS, which is suitable for high radiation spacecraft environments. We are currently reimplementing the architecture in three-micron bulk CMOS and will be finished in the second quarter of 1987. Papers describing the full details of the processor and architecture have been submitted to the 1986 FORML Conference.

Martin E. Fraeman
John R. Hayes
Robert L. Williams
Thomas Zaremba
Johns Hopkins University
Laurel, Maryland

Practical Considerations for
Floating-Point Arithmetic

Richard Wilton
Marina del Rey, California

In most high-level languages, whether or not to use floating-point arithmetic is not even a question. Fortran, PL/1 or C programmers simply take for granted that when they wish to compute with real numbers, the language they are using offers the tools to do so. The presence of arithmetic data types in such high-level languages allows the selective use of integer or real arithmetic.

In contrast, Forth deals with objects on a somewhat less abstract level. A Forth programmer must always be aware of the low-level representation of real numbers and the manner in which arithmetic operators are implemented. These considerations are much less important to programmers in most high-level languages.

This article discusses some of the practical points involved in doing Forth floating-point arithmetic. It starts by covering the salient low-level features of floating-point system design in Forth. The simple source code examples which follow illustrate some of the points to consider in designing real arithmetic into a Forth application.

Real-Number Representation

One of the first questions the implementor of floating-point numbers has to solve is that of the representation of real numbers. The usual representation is a simple data structure containing an exponent (sometimes called the "characteristic"), a significand ("mantissa") and a sign bit. An example is shown in Figure One.

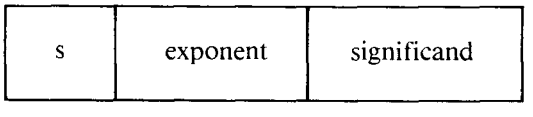


Figure One

With an eight-bit exponent, a twenty-three-bit significand and one sign bit, this real-number representation could be stored in two sixteen-bit words on the usual Forth stack. Many similar representations can be used in Forth floating-point implementations.

A Forth systems programmer chooses the representation best suited to a particular hardware and software situation. For example, some representations are more easily used in software floating-point primitives, whereas others correspond to the representation used by a floating-point coprocessor such as the AMD 9511 or the Intel 8087, or to that used by firmware routines such as those in the Apple Macintosh or in the IBM PC's BASIC ROM.

A Forth application programmer who uses floating-point arithmetic must be aware of the representation used, because the dynamic range and accuracy of real numbers is implicit in their representation. Also, if you wish to manipulate real numbers with standard Forth operators such as **2@** or **CMOVE**, you must know how many bytes of storage are required for each real number.

Manipulating Real Numbers

Another important point to consider when you use floating-point arithmetic in Forth is the problem of where to place real numbers so that they can be manipulated conveniently. Because integer arithmetic is sufficient for Forth's memory-conserving, threaded code interpreter, the Forth virtual machine is implicitly biased towards performing integer arithmetic. Integrating real numbers and floating-point operators into the standard Forth system thus demands careful consideration.

There are two common solutions to this problem. One is to maintain real numbers on Forth's parameter stack. The other is to design a separate real-number stack which is tightly integrated into the standard Forth interpretive system. Both approaches are viable.

Using the Parameter Stack. For most purposes, there is no reason to avoid placing real numbers on the parameter stack, even though they are almost certainly represented as thirty-two-bit, forty-eight-bit or even sixty-four-bit numbers. After all, the usual Forth stack is already cluttered with data items of various sizes and types, including eight-bit characters, sixteen-bit signed and unsigned integers, thirty-two-bit integers and addresses of various sizes.

An advantage to manipulating floating-point data on the Forth parameter stack is that the usual stack and memory operators can be easily adapted to handling real numbers. For instance, if a real number is represented in sixty-four bits, then

```
: FDROP ( r -- )
DROP DROP DROP DROP ;
```

is exactly analogous to **DROP** for sixteen-bit integers or to **2DROP** for thirty-two-bit integers. Similar operators, such as **FDUP**, **FSWAP**, **FPICK** and so on can be defined in terms of the standard Forth stack words.

A common problem is that the parameter stack can quickly become crowded, particularly when sixteen-bit integers and addresses must be maintained on the stack at the same time as real numbers. Bugs introduced by inaccurate stack operations (for example, **SWAP** instead of **FSWAP**) can be notoriously difficult to track down.

Using a Separate Stack. In an effort to avoid stack clutter, some implementors of Forth floating-point support simply maintain all real numbers on a separate, dedicated stack. This design makes life much easier for programmers who make heavy use of the parameter stack.

The separate stack approach can also lead to significantly improved performance if it is supported in hardware. For example, the Intel 8087 arithmetic coprocessor maintains its own stack. (The stack is only eight deep, but this is sufficient for most applications.) A separate real-number stack thus maps directly onto the hardware, which simplifies the low-level software primitives and leads to increased execution speed in application programs.

In practice, neither approach to floating-point stack design has proved to be unequivocably better. Other considerations, including source code readability, portability and the asymmetry of floating-point hardware with standard Forth system design, lead to compromises in system complexity and in execution speed.

```
( STEST -- Scaled arithmetic version )
: AREA ( radius -- area )
   DUP *                                              \ r^2
   355 113 */ ;                                   \ pi * r^2

( USTEST -- Unsigned scaled arithmetic version )
: AREA ( radius -- area )
   DUP *                                              \ r^2
   355 UM* 113 UM/MOD SWAP DROP ;                 \ pi * r^2

( FTEST -- Floating point version )
: AREA ( radius -- area )
   DUP M* D>F  FPI F* ;                           \ pi * r^2

( F87TEST -- version which uses 8087 stack )
: AREA ( radius -- area )
   0              \ convert to double (8087 "short integer")
   IS>AP APDUP (FMULP)                    \ r^2 on 8087 stack
   (FLDPI) (FMULP) AP>FL ;                        \ pi * r^2


( Timing loop )
: TEST ( -- )
   !TIMER
  100 0 DO
   101 1 DO  I AREA DROP  LOOP          \ substitute FDROP in ..
        LOOP                       \ .. floating point versions
   .TIMER ;
```

Table One. Source Code Examples.

STEST	USTEST	FTEST (SFP)*	FTEST (8087)	F87TEST
5.16	3.46	75.63	5.88	3.63

Table Two. Timings for 10,000 executions of AREA (IBM PC, 4.77 MHz 8088).

STEST	USTEST	FTEST (SFP)*	FTEST (8087	F87TEST
1.16	0.71	18.34	2.26	1.53

Table Three. Timings for 10,000 executions of AREA (IBM PC AT, 8 MHz 80286).

*SFP means "Software Floating Point."

Floating-Point Operators

Most programmers perform floating-point arithmetic in Forth with operators that are analogs of the standard Forth integer arithmetic operators. Floating-point operators with analogous names (e.g., **F+**, **FDUP**, **F@**) perform functions analogous to the standard integer operators. It is easy to program "intuitively" with this type of system.

Some programmers prefer to redefine the standard integer operators so that they work with real numbers instead. These redefined operators are maintained in a separate vocabulary. This approach allows a given piece of source code to be used with either number type, simply by switching vocabularies. Also, the same set of operators can be used for either integer or real arithmetic, just as they are in Fortran and other high-level languages.

The disadvantages of both approaches are clear. Using a parallel set of operators adds two or three dozen new words to a language which already demands familiarity with several hundred words. However, redefining existing Forth integer operators to handle real numbers also creates problems. A program which manipulates both data types simultaneously soon becomes littered with vocabulary changes which obscure the functional meaning of the source code.

Other Considerations

Forth systems programmers must consider many other issues of floating-point implementation, including accuracy, rounding, representation of values which cannot be exactly expressed in binary, infinity, error trapping (division by zero, invalid arguments to trigonometric functions) and so on. Such implementation details are often irrelevant to an application programmer. However, in many instances, knowledge of the exact behavior of the floating-point package is critical to debugging as well as to obtaining accurate results.

A Simple Example

At this point it is worthwhile to examine some source code. Apart from superficial differences in notation, it is important to observe the implicit differences between integer and floating-point arithmetic when each is used for computation of fractional quantities. Although there are applications which by nature demand the use of either integer or real arithmetic, situations frequently arise in which the choice is affected by stylistic or performance considerations.

The simple example in Table One calculates the area of a circle four different ways. The first two, **STEST** and **USTEST**, use scaled integer arithmetic. The value for pi is the well-known ratio 355/113, which is accurate to six decimal places. The scaling in **USTEST** looks slower but runs faster because it does not use **/** and thereby avoids the overhead of floored division.

The second pair of examples, **FTEST** and **F87TEST**, use floating-point arithmetic to do the same work. **FTEST** is written with a set of floating-point operators which parallel the usual integer operators. It uses the Forth parameter stack for all real arithmetic, so integers and real numbers coexist on the stack at the same time. The last example, **F87TEST**, uses the Intel 8087's separate stack to hold real numbers for intermediate calculations.

A comparison of the source code reveals little on the surface apart from the somewhat obscure operators used to manipulate the 8087 stack directly. There is, however, a great deal of difference in dynamic range and in precision implied by the use of floating-point operators. Any increase in precision of the integer versions **STEST** and **USTEST** would require additional scaling operations with a significant performance degradation as a consequence, as well as additional code required to support scaling.

Tables Two and Three contain typical performance data. Most of the differences in timing between the examples is due to the time required for multiplication by pi. The timing loop calls the **AREA** routine 10,000 times and uses the computer's system clock (accurate to about 0.06 seconds on an IBM PC) as a timer.

The poor performance of **FTEST** when real arithmetic is carried out in software (SFP) stands out in sharp contrast to the other results. (Nevertheless, it is still a bit faster than interpreted BASIC!) What is striking is that the speed of floating-point arithmetic using a hardware coprocessor is quite close to that of integer arithmetic, yet the degree of precision and dynamic range achievable with the use of floating-point arithmetic is far beyond the capabilities of integer arithmetic, scaled or not.

Practical Experience

It would be wrong to extrapolate from these simple timing data that real arithmetic will always be just about as fast as integer arithmetic in Forth. The point is that the performance penalty for using floating-point arithmetic in Forth is negligible in situations where an application demands precision and dynamic range. There is no reason to use scaled arithmetic to avoid decreased run-time performance if the degree of performance degradation is not critical and if significantly increased source code complexity results.

This observation has been thoroughly demonstrated in real-world situations. Floating-point Forth programs have been successfully utilized in applications such as high-level display graphics, real-time engineering telemetry processing and industrial quality-control analysis. A Forth program which uses floating-point arithmetic is often the best approach to an application which demands real-number processing as well as interactive hardware control.

With inexpensive, widely available floating-point hardware, real numbers can be handled in a sophisticated manner without sacrificing either speed or the many conveniences of the standard Forth interpretive environment. Furthermore, in well-integrated systems such as the Apple Macintosh, it behooves a Forth programmer to take advantage of readily available firmware support for real arithmetic. With a critical eye to the factors described in this article, you can easily integrate floating-point arithmetic into Forth applications.

Screenless Forth

Carl A. Wenrich
Tampa, Florida

Don't get me wrong: I love my Laxen & Perry F83 package. It is the most elegant piece of code I've seen since the last thing I wrote myself. But, somehow, I've never been able to get to the point where I actually enjoy screen editing. Even with everything that's done to help, I still find it tedious.

On the other hand, editing with my **SEE** editor (C Ware Corporation, P.O. Box C, Sunnyvale, CA 94087) is a pure joy. So to have my cake and eat it too, I wrote this little piece for my IBM PC to escape the tyranny of the silent screen. It allows you to create source modules using any ASCII text file editor (even DOS's EDLIN, if you're desperate).

Here's how it works. F83 is set up with four disk buffers of 1024 bytes each at the top of memory. I just redefined that space as a 4K source file buffer. Any programs larger than 4K can be broken down into 4K modules and chained together easily.

Let's take a look at the commands required to implement this screenless Forth system. As you can see by glancing at the listing, there really isn't very much to it. What we have is yet another indication of the power of Forth: you can do quite a lot with very little.

Since some of the new words are duplicates of existing commands, we begin by defining a new vocabulary named **UNSCREEN** to keep them separate. **B/FILE** is the variable that will hold the number of bytes in whatever source file we load. **MOD-BUF** is the address of the 4K buffer at hex F000 where the file will go.

REC-SIZE and **FILE-SIZE** serve as offsets into the file control block; they leave the record-size and file-size addresses, respectively. **OPEN-FILE** is similar to the existing **OPEN-FILE** command, except this one checks to see that the source file is no larger than 4K. If it is, we abort with an appropriate error message; if it isn't, we store the number of bytes in **B/FILE**.

READ-CHAR reads one character from the source file. **READ-SEQ** is the command that reads a sequential source file into the 4K buffer at **MOD-BUF**. The record size is set to one so that the file you need is the file you get. The DTA (data transfer address) is set up at **PAD**. Each time a character is brought in, it

```
     1
 0 \ LOAD BLOCK                                          05APR86CW
 1
 2   ONLY FORTH ALSO DEFINITIONS
 3
 4   WARNING OFF
 5
 6 : NLOAD .S (LOAD) ;    ' NLOAD IS LOAD
 7
 8   2 4 THRU
 9
10
11


     2
 0 \ UNSCREEN REC-SIZE FILE-SIZE OPEN-FILE READ-CHAR     05APR86CW
 1
 2   VOCABULARY UNSCREEN
 3   ONLY FORTH ALSO DOS ALSO UNSCREEN DEFINITIONS
 4   VARIABLE B/FILE   61440 CONSTANT MOD-BUF
 5
 6 : REC-SIZE (S adr -- adr' ) 14 + ;
 7 : FILE-SIZE (S adr -- adr' ) 16 + ;
 8
 9 : OPEN-FILE (S -- ) IN-FILE @ DUP 15 BDOS DOS-ERR?
10   ABORT" Open error"  FILE-SIZE @ DUP 4096 >
11   ABORT" File over 4k"  B/FILE ! ;
12
13 : READ-CHAR (S -- ) IN-FILE @ 20 BDOS DOS-ERR?
14   ABORT" Read error"  ;
15


     3
 \ READ-SEQ (LOAD) (SOURCE)                              05APR86CW

: READ-SEQ (S -- ) IN-FILE @ DUP REC-SIZE 1 SWAP !
  FILE-SIZE @ 0 DO
    READ-CHAR PAD C@ BL MAX MOD-BUF >IN @ + C! 1 >IN +!
  LOOP ;

: (LOAD) (S -- ) ?DEFINE !FILES OPEN-FILE >IN OFF
  PAD SET-DMA READ-SEQ >IN OFF BLK ON RUN ;

: (SOURCE) (S -- adr len ) BLK @ IF
    MOD-BUF B/FILE @ ELSE TIB #TIB @
  THEN ;


     4
\ (?ERROR)                                               05APR86CW

: (?ERROR) (S adr len f -- ) IF
    TYPE CR SP0 @ SP! PRINTING OFF BLK @ IF
      CR MOD-BUF >IN @ BOUNDS DO I C@ EMIT LOOP
    THEN QUIT
  ELSE
    2DROP
  THEN ;

' (LOAD) IS LOAD   ' (?ERROR) IS ?ERROR   ' (SOURCE) IS SOURCE
```

is compared to **BL**. Printable characters are transferred to **MOD-BUF** and control characters are converted to blank spaces.

(LOAD) fires up the interpreter after the file has been read into memory. It combines the functions of the normal **OPEN** and **(LOAD)** commands. After **LOAD** is revectored to the **UNSCREEN** version of **(LOAD)**, all you have to do is type "**LOAD** filename.ext" and the file will be opened, read into memory and interpreted.

If there are no detectable errors in the source file, you will receive the all-familiar "ok" from the interpreter. Of course, you will have to revector **LOAD**, **SOURCE** and **?ERROR** back to **FORTH** vocabulary versions if you want to play with screens for any reason.

Any detectable source file error will trigger a memory dump from the first byte of the source file buffer **MOD-BUF** to the end of the offending word. This will let you know exactly where the error was found. If a standard message is associated with the error, it will be displayed as well.

(SOURCE) is a slightly modified version of same. **BLK** is now used as a flag which indicates whether the input stream is coming from the keyboard or from the module buffer. **MOD-BUF** supplies the address, and **B/FILE** supplies the number of bytes to be interpreted.

?ERROR is again a modification of the **FORTH** vocabulary's version. But instead of leaving parameters for the **WHERE** command, it dumps the module buffer up to and including the word that triggered the abort. Of course, if you happen to be interpreting from the keyboard, it just flags the error as before.

The only thing left to do now is revector **LOAD**, **SOURCE** and **?ERROR**. Once this is done, you had better not try any screen manipulations unless you first revector back to the **FORTH** versions, because you will probably crash.

But now you are free to load one or more ASCII text files and they will be interpreted just as though they were screen files. To demonstrate how this is done, and how easily files can be chained, here's a little sample session. It assumes that three files of Forth code have already been created. It also assumes that the last two lines of code in **FILEA.BLK** look like this:

```
CR .( LOAD FILEB.BLK )
LOAD FILEB.BLK
```

and that the last two lines of code in **FILEB.BLK** look like this:

```
CR .( LOAD FILEC.BLK )
LOAD FILEC.BLK
```

Now, assuming that the **UNSCREEN** definitions have been loaded, all you have to do is type **LOAD FILEA.BLK** and wait. If the files are large (near 4K), it will go down something like this:

The selected drive will come on and **FILEA.BLK** will be read into memory. After the drive goes off, it will seem as though nothing is happening. Actually, the file is now being interpreted. As soon as the interpreter gets to the end of **FILEA.BLK** you will see **LOAD FILEB.BLK** appear on the screen and the drive will come on again. **FILEB.BLK** will now be read in and interpreted. **LOAD FILEC.BLK** will then appear, and **FILEC.BLK** will be read in and interpreted.

At this point, you are ready to run your application. You may leave your image by entering "**SAVE-SYSTEM** filename.com" and boot right into it by entering "' program **IS BOOT**".

In any case, I think you will find that editing source modules will become a bit more enjoyable. And as an added bonus, you will find they take up a great deal less disk space — screens are notorious disk hogs because of all the white space they require. As a result, you will probably be more likely to structure (indent) your Forth source code the way it was intended, instead of squeezing it into that 16x64 box like most of us.

Tracking the Beast

Nathaniel Grossman
Los Angeles, California

Humankind has been fascinated by numbers throughout all of its recorded history. To its one hand lay mathematics, with the abstract theory of numbers and more exotic developments. We have cuneiform evidence that the Pythagorean Theorem was known to the ancient Babylonians, and abundant testimony from the Greek tradition of a feverish devotion to the study of integers. At its other hand lay the pseudo-scientific (as we now call it) numerology, with ample evidence beginning with the oldest surviving literary texts to show that this study of the influence of numbers upon human affairs developed parallel to and, sometimes, hand-in-hand with the scientific study of numbers[1]. Up to recent times, certain numbers or combinations of numbers were thought to have special significance for humanity or for particular humans. Many, if not most, of us retain traces of these ancient superstitions, no matter how rational we deem ourselves to be[2].

Religious writings are a fertile source of numerological lore. The Bible is no exception, as Hooper fully illustrates[1]. Biblical numerology has been developed in both the Jewish and the Christian traditions. The Jewish Kabbalists refined Old Testament numerology into the real-time numerological art of *gematria*. Early Christian numerology developed gematria-like techniques based upon the fact that the letters of the Greek alphabet, like those of the Hebrew, carried dual meanings as numbers.

Perhaps the most notorious numerological passage in the Bible occurs in the New Testament, in the Book of Revelation of St. John the Divine (13.18):

Here is wisdom. Let him that hath understanding count the number of the beast: for it is the number of a man; and his number is Six hundred threescore and six.

The Beast has been considered from early Christian times to be an apocalyptic enemy of mankind, the Antichrist. Original numerological attempts to identify The Beast with a historical man produced various candidates. Most prominent among these is the Roman emporer Nero, the calculation being based on the values of the letters in the Greek alphabet, in which the earliest available versions of the New Testament were written. During the medieval period, calculations in Roman numerals were common. Also, in more recent times, attempts were made to pin the label of The Beast on contemporary persons such as Martin Luther. In our time, the likes of Franklin Delano Roosevelt were Beastified by their enemies.

These identifications seem not to have had much influence on human affairs, but they may have conferred some benefits on their devisers. We recognize nowadays that it is more desirable to break pencil points than heads. Next time you feel compelled to take up the cudgel, use the boot instead: boot up this program and, with its help, identify your adversary as The Beast.

Type in the twelve screens. When you execute **1 LOAD**, a startup message will appear on your display with instructions on how to begin. Follow the prompts. Figure One shows how one session went. I loaded the program and read the prompt, then executed **BEAST?**. Responding to the prompt, I entered the name of a friend, Ignia Incendiari. When I pressed the carriage return, I was asked whether the calculation should make the special identifications **U->V**, **Y->I** and **W->VV** (the calculation proceeds, medieval style, in Roman numerals). I answered yes, whereupon the numerical value of the name was analytically displayed. The value was 605. (At this point in the program, I would have liked to put in a whistle to inform me whether the total was greater than or less than 666 and by how much, but I decided to keep to twelve screens in order to print the program on two pages.) A bit of pencil work showed me that I was sixty-one short. The additional letters D and C were therefore barred, but various combinations of L, X, V and I were available. A few mystic passes over these letters, Scrabble-style, and Lisa appeared, so I responded to the prompt "Another name?" with yes and entered the fuller name, Ignia Lisa Incendiari. It was clear that the remaining deficiency was ten. Now I realized that my friend had withheld her middle name; she is Ignia Alexis Incendiari. When I entered her full name, the proclamation came back to me: her number is the number of The Beast. That satisfied me, and I told the program that I was through with it. But wait! You ask: How can I assume that her name is Ignia *Alexis* Incendiari? Isn't that fudging? I am forced to admit that, indeed, I have fudged, but in doing so I am only following the lead of the great numerologists of the past, who fudged mightily. And perhaps I have discovered a truth that is unknown even to Ignia Incendiari herself. Numbers don't lie — or do they?

References

1. Hopper, Vincent Foster. *Medieval Number Symbolism*. Cooper Square Publishers, New York, 1969. The word "medieval" in the title does not disqualify this book as a reference on current-day numerology.
2. Bell, Eric Temple. *Numerology*. Baltimore, 1933. Reprinted 1981. This little book, scarce but well worth finding to read, was written by an eminent mathematician who also published popular science fiction under the pseudonym "John Taine."

```
1 load


...........

This program will help your calculations toward
identifying the Beast of Revelation:

   Here is wisdom. Let him that hath understanding
count the number of the beast: for it is the number
of a man: and his number is Six hundred threescore
and six.  -- The Revelation of St. John the Divine, 13:18


Type  BEAST? <return>  to begin.



BEAST?

Type a 'name' of no more than 80 characters,
including spaces and upper and lower case letters,
then press <return>.

Ignia Incendiari

Shall U&u (vee), Y&y (eye), and W&w (two vees) be counted?
(Y!N): y


Ignia Incendiari:

  0 #Ms =    0
  1 #Ds =  500
  1 #Cs =  100
  0 #Ls =    0
  0 #Xs =    0
  0 #Vs =    0
  5 #Is =    5
          ----
           605

Another name? (Y!N): y

Type a 'name' of no more than 80 characters,
including spaces and upper and lower case letters,
then press <return>.

Ignia Lisa Incendiari

Shall U&u (vee), Y&y (eye), and W&w (two vees) be counted?
(Y!N): y

Ignia Lisa Incendiari:

  0 #Ms =    0
  1 #Ds =  500
  1 #Cs =  100
  1 #Ls =   50
  0 #Xs =    0
  0 #Vs =    0
  6 #Is =    6
          ----
           656
```

```
Another name? (Y!N): y

Type a 'name' of no more than 80 characters,
including spaces and upper and lower case letters,
then press <return>.

Ignia Alexis Incendiari

Shall U&u (vee), Y&y (eye), and W&w (two vees) be counted?
(Y!N): y


     The number of Ignia Alexis Incendiari is 666,
     the number of The Beast of the Book of Revelation!

Another name? (Y!N): n


     DONE!
```

```
SCR#   1
\ The Beast of Revelation: loader screen   Forth 83  NB 04/18/86

MARK   ( drop loaded program by FORGET MARKER )

2 12 THRU   \ 1 LOAD puts The Beast into the dictionary!

\ Startup message
DARK  ( clear screen ) BEEP ( capture user )  CR CR
.( This program will help your calculations toward ) CR
.( identifying the Beast of Revelation: ) CR CR
.(    Here is wisdom. Let him that hath understanding ) CR
.( count the number of the beast: for it is the number ) CR
.( of a man: and his number is Six hundred threescore ) CR
.( and six.  -- The Revelation of St. John the Divine, 13:18 )
CR CR CR CR  .( Type  BEAST? <return>  to begin. ) CR CR CR CR


SCR#   2
  0 \ Registers and buffers                              NB 04/18/86
  1
  2 \ Registers to hold occurence counts for Roman-numeral letters
  3 VARIABLE #M   VARIABLE #D   VARIABLE #C   VARIABLE #L
  4 VARIABLE #X   VARIABLE #V   VARIABLE #I
  5
  6 \ Buffer to hold pattern text -- 80 chars max, including spaces
  7 CREATE NAME   80 ALLOT
  8
  9 VARIABLE SPAN?   \ Alias for SPAN
 10
 11 : READNAME
 12 \ Accept name from keyboard and move to NAME-buffer
 13     CR  NAME 80  EXPECT  SPAN @ SPAN? !   ;
 14
 15
```

```
SCR#   3                                                  SCR#   4
\ Characters that are Roman numerals          NG 04/18/86  0 \                                          NG 04/18/86
                                                           1
\ Stack the ASCII chars:   (   --- UC lc )                 2 : DDUP   ( n --- n n n )   DUP DUP   ;
: ULMm   ASCII M  ASCII m   ;                              3
: ULDd   ASCII D  ASCII d   ;                              4 : ?ASCII=   ( n n n1 n2 --- f )
: ULCc   ASCII C  ASCII c   ;                              5 \ True if n = n1 or n = n2; the n's will be ASCII characters
: ULLl   ASCII L  ASCII l   ;                              6     ROT =  >R  =  R>  OR   ;
: ULXx   ASCII X  ASCII x   ;                              7
: ULVv   ASCII V  ASCII v   ;                              8 : 1+!   ( addr ---   ) \ add 1 to contents of addr
: ULIi   ASCII I  ASCII i   ;                              9     1 SWAP +!   ;
: ULUu   ASCII U  ASCII u   ;                             10
: ULJj   ASCII J  ASCII j   ;                             11
: ULYy   ASCII Y  ASCII y   ;                             12 : INIT-#REGS   \ Initialize the count registers to 0
: ULWw   ASCII W  ASCII w   ;                             13     0 #M !  0 #D !  0 #C !  0 #L !  0 #X !  0 #V !  0 #I !   ;
                                                          14
                                                          15

SCR#   5                                                  SCR#   6
\ Fudge for extended Roman numerals           NG 04/18/86  0 \ Count occurences of the numerals              NG 04/18/86
                                                           1
\ The next variable is true if the fudges                  2 : ?ROMAN#   ( n --- n )   \ count an occurence of a roman #
\ J and Y to I, U to V, W to V+V are on.                   3     DDUP ULMm ?ASCII=  IF  #M 1+!  ELSE
VARIABLE #FUDGE                                            4     DDUP ULDd ?ASCII=  IF  #D 1+!  ELSE
                                                           5     DDUP ULCc ?ASCII=  IF  #C 1+!  ELSE
: ?FUDGE   (   --- f )                                     6     DDUP ULLl ?ASCII=  IF  #L 1+!  ELSE
    #FUDGE @   ;                                           7     DDUP ULXx ?ASCII=  IF  #X 1+!  ELSE
                                                           8     DDUP ULVv ?ASCII=  IF  #L 1+!  ELSE
: FUDGE    ( n f ---   )   \ count occurences of fudged num'ls  9     DDUP ULIi ?ASCII=  IF  #I 1+!  ELSE
    IF                                                    10                        THEN THEN THEN THEN THEN THEN THEN   ;
      DDUP ULUu ?ASCII=  IF  #V 1+!  ELSE                 11
      DDUP ULYy ?ASCII=  IF  #I 1+!  ELSE                 12 : ?EXTENDED_ROMAN#   ( n --- n )   \ also count U, Y, W
      DDUP ULWw ?ASCII=  IF  2 #V +! ELSE - \ W is two vees 13     ?ROMAN#
                         THEN THEN THEN                   14     DUP  ?FUDGE ( u,y,w too? )   IF  FUDGE  THEN   ;
    THEN   ;                                               15


SCR#   7                                                  SCR#   8
\ Values of numerals, calc value of name      NG 04/18/86  0 \ Letter discriminators                        NG 04/21/86
                                                           1 : .PATTERN_MESSAGE   (   ---   )
: PATTERN-SCAN   (   ---   )                               2     CR   ." Type a 'name' of no more than 80 characters, "
    SPAN? @ 0 DO          \ for each character in the pattern  3     CR   ." including spaces and upper and lower case letters, "
      NAME I + C@         \ fetch it from the NAME-buffer   4     CR   ." then press <return>. "   CR   ;
      ?EXTENDED_ROMAN#    \ count it if a roman numeral     5
      DROP                \ discard the character           6 : YES?   ( char --- f )   \ True only if one of Y or y
    LOOP   ;                                               7     DUP  ASCII Y =  SWAP  ASCII y =  OR   ;
                                                           8
: NUMBER?   (   --- n )                                    9 : NO?    ( char --- f )   \ True only if one of N or n
\ cumpute value of name from counts stored by the scan    10     DUP  ASCII N =  SWAP  ASCII n =  OR   ;
    0  #M @ 1000 * +  #D @ 500 * +  #C @ 100 * +  #L @ 50 * +  11
       #X @   10 * +  #V @   5 * +  #I @       +   ;          12 : YES_OR_NO?   ( char --- f )   DUP  YES?  SWAP  NO?  OR   ;
                                                          13
                                                          14 : Y!N_MESSAGE   CR CR  BEEP
                                                          15     ." You must respond with Y or N, then <return>!"  CR   ;
```

(Continued on page 27.)

FORTH INTEREST GROUP MAIL ORDER FORM

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

MEMBERSHIP

IN THE FORTH INTEREST GROUP

108 - MEMBERSHIP in the FORTH INTEREST GROUP & Volume 8 of FORTH DIMENSIONS. No sales tax, handling fee or discount on membership. 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 publication discounts, group health and life insurance, an on-line data base, a large selection of Forth literature, and many other services. Cost is $30.00 per year for USA, Canada & Mexico; all other countries may select surface ($37.00) or air ($43.00) delivery.

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 which entitles you to a 10% discount on publications from FIG. Your member number will be required to receive the discount, so keep it handy.

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 _____

ALL 7 VOLUMES $75.00 **SAVE $35.00**

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
N E W (Dr. Dobb's) . . . . . $23/25/35 _____
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
LMI PC/FORTH 3.0
MasterFORTH 1.0
TaskFORTH 1.0
PolyFORTH® II

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.

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
108	MEMBERSHIP				SEE BELOW
	★ HOLIDAY SPECIALS ARE GOOD ON ORDERS FOR CANADA, MEXICO & USA ONLY. EUROPE ADD 40% AUSTRALIA AND ASIA ADD 60%. SORRY — MEMBER DISCOUNTS ARE NOT APPLICABLE WITH HOLIDAY SPECIALS. ★				

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

☐ VISA ☐ MASTERCARD

Card # _____

Expiration Date _____

Signature _____

($15.00 minimum on charge orders)

SUBTOTAL	
10% MEMBER DISCOUNT	
SUBTOTAL	
CA. RESIDENTS SALES TAX	
HANDLING FEE	**$2.00**
MEMBERSHIP FEE ☐ NEW ☐ RENEWAL $30/37/43	

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%.

A Simple Translator:
Tinycase

Allen Anway
Superior, Wisconsin

I recently wrote several menu-driven programs and observed the following: frequently, the operator must press a key for the desired response, but the programmer wants a value output other than that of the pressed key. Thus, the programmer must translate an arbitrary ASCII keystroke into another arbitrary number. If programmed once, the **CASE** structure is a good solution because of its clear, easy-to-change structure. If programmed often, **CASE** and all of its branches consume quite a few bytes.

So I wrote the compact **TINYCASE** to inspect a similarly ordered array of sixteen-bit numbers for matches and to output to the stack the translated number when a match is found. If no match is found, it outputs a default number, just as can be done in **CASE**. Screens 80 and 81 show **TINYCASE** implemented both in high-level Forth and in **;CODE** assembler. The high-level **BEGIN ... WHILE ... UNTIL** construction comes from the remarkable article by Harralson (*Forth Dimensions* VI/2). It takes some stack gymnastics for the high-level word to work out, so the **;CODE** word is much preferred both for reasonable compactness and for blazing speed.

Screen 83 shows identical examples of **TEST1** and **TEST2** with stack effects of (#entered -- #result). One must tell **TINYCASE** in advance how many groups there will be, four in this case. One does not have to put in a default value, negative twelve in this case. But lacking such only means that if one enters the **TINYCASE** default condition, one most likely will get part of the header of the next word in the dictionary. **CASE** must explicitly have a default or no other number will be put on the stack.

Both **TEST1** and **TEST2** operate as follows:

```
2     TEST1 .     234    ok
3     TEST1 .     -12    ok
97    TEST1 .     979    ok
```

```
SCR # 80
  0 ( # 080 ) ( TINYCASE program ) FORTH-83
  1 ( Allen Anway, UW-Superior 4-1-85 )
  2
  3 : TINYCASE  CREATE 4 * 2- , DOES>
  4
  5 ( #entered\pfa )
  6
  7     DUP @ >R -4
  8
  9     BEGIN
 10
 11       6 + 2DUP + @ 3 PICK   -
 12
 13     WHILE
 14
 15       2-   DUP  R@          U>
 16
 17     UNTIL
 18
 19     RDROP 2+ + @ SWAP DROP ( #result ) ;
 20
 21 ;S
 22 compile  ( #-of-tests --- )
 23 execute  ( #entered --- #result )

SCR # 81
  0 ( # 081 ) ( TINYCASE  program )
  1 HEX
  2
  3 : TINYCASE  CREATE 4 *    , ;CODE
  4
  5   2 .# LDY, W )Y LDA, N STA,  INY,
  6   BEGIN,                      INY,
  7     W )Y LDA,     INY, BOT    CMP,
  8     ZS IF,  ( <BNE> type of branch   )
  9       W )Y LDA,        BOT 1+ SBC,
 10     THEN,
 11     06F0 ,  ( branch )  INY, INY, N CPY,
 12   CS UNTIL, ( <BCC> type of branch   )
 13   ( branch here from F0 06 <BEQ +06> )
 14                               INY,
 15   W )Y LDA,      PHA,         INY,
 16   W )Y LDA, PUT JMP,          END-CODE
 17
 18 DECIMAL
 19
 20 -->
 21
 22 compile  ( #-of-tests --- )
 23 execute  ( #entered --- #result )

SCR # 82
  0 ( # 082 ) ( TINYCASE  example )
  1
  2 4 TINYCASE TEST1        2  , 234 ,
  3                         7  , 789 ,
  4                        18  , 181 ,
  5                        97  , 979 ,
  6                            -12 ,
  7
  8
  9 ( 30 bytes of code total, 10 of header )
 10
 11
 12
 13          : TEST2 CASE  2 OF 234 ENDOF
 14                        7 OF 789 ENDOF
 15                       18 OF 181 ENDOF
 16                       97 OF 979 ENDOF
 17        ( alternately DUP OF -12 ENDOF )
 18                            -12 SWAP
 19               ENDCASE ;
 20
 21 ( 77 bytes of code total, 10 of header )
 22
```

Classes in Forth

Vince D. Kimball
Ipswich, Massachusetts

If one wishes to do object-oriented programming in Forth, one must first add the class concept to the language. A Forth-like solution to the problem, a minor modification of the vocabulary concept, is proposed.

Overview

The principles of transparency and localization seem to be central to the current interest in object-oriented programming. Transparency emphasizes the wish to use generic operators across data structures, and localization emphasizes the desire to partition a group of data structures and operations upon them into a separate entity which may be understood more or less on its own. Currently, Forth does not seem to support these principles in any direct way. Multiple-code-field words are a first step toward generic operators, but they are flawed for general use in that they do not allow adding to the original class of operators to be used with a given data structure. They are useful, however, for the very basic operators which are common to most data structures. Vocabularies seem to provide localization, but at present they are insufficient to the task because they do not allow easy mixing of different vocabularies or the explicit specification of linkages among vocabularies.

If we accept these principles as useful but want to retain the flexibility and performance of Forth, we must discover how to add structures to Forth to support them without making Forth into a pale echo of Smalltalk. The proposed solution is to implement the class as a modified vocabulary and to enable the use of the class name as a prefix operator for modifying the dictionary search sequence. I believe that this unique concept will provide the power of object-oriented programming without sacrificing any of Forth.

Plan

An extremely simple method of adding classes to Forth involves the use of Forth's built-in vocabulary system as a foundation. The addition of six new words plus a modification of Forth's dictionary lookup sequence will provide the core of object programming while maintaining the idiom and flexibility of Forth. The first three new words **CLASS**, **CLASS@** and **<SUPER** allow for the definition of classes. The last three new words **CLASSVAR**, **DEFER** and **CLASS>** provide the useful ability to defer binding the name of a class to a word until run time. Other words may suggest themselves as more experience with this style of programming is gathered.

Class Definition Words

Classes would be defined according to the following form:

CLASS ClassName
<SUPER SuperClassName
CLASS@ ClassName **DEFINITIONS**
(definitions in class ClassName)
FORTH DEFINITIONS

The word **CLASS** would create (in the compilation vocabulary) a dictionary entry for ClassName which specifies a new list of word definitions forming the class being defined. Subsequent execution of ClassName will be as a prefix operator making the words in the class the first part of the search order during the next dictionary lookup. Thus, the phrase "ClassName WordName" would find the word WordName in the class ClassName, if there was one, and the search order would be the same after the phrase as it was before it. The word **<SUPER** would be used to indicate the superclass of the class just defined. It would chain the class indicated by ClassName to the class indicated by SuperClassName. When a dictionary search of ClassName is exhausted, SuperClassName would be searched. Those classes without superclasses could be declared as

CLASS ClassName **<SUPER** Object

The Object class would be the primary class, holding definitions common to all classes. Classes defined without using the **<SUPER** word would not be chained to any superclass, which might be useful in some cases. The word **CLASS@** would be used in the phrase "**CLASS@** ClassName" to make the following class name the first vocabulary in the regular search order, rather than the active class as it normally is.

Class Variables and Deferred Binding

As defined above, the class of an object must be known when the word involved is defined. In some cases it may be convenient not to have to specify the name of a class in advance. This ability is provided by employing the following phrase:
ClassVarName **DEFER** WordName

When this phrase is executed, WordName is looked up in the vocabulary in the class which is currently referenced by ClassVarName and then is executed. This lookup will take a certain amount of time, but the increase in flexibility may be worthwhile at times. It would be an error if WordName is undefined at run time, of course.

Class variables are defined by using the standard form:

CLASSVAR ClassVarName

This phrase would define a null class variable which would have to be assigned a real class to be of use. Unlike classes, class variables are not considered prefix operators because they execute at run time to provide information to **DEFER**. The method of assigning a class to a class variable had perhaps be best left to the discretion of the implementor, although the following form may be satisfactory:

CLASS> ClassName ClassVarName

The difficulty in implementing this operation is ensuring that ClassName is not executed as a prefix operator.

Dictionary Lookup

The final change to Forth to provide classes would be to modify its dictionary lookup sequence in order to enable the use of class names as prefix operators which modify the search order only on a temporary basis. The implementation of this new lookup sequence would seem to require that there be an **ACTIVECLASS** vocabulary to be searched before **CONTEXT** and **CURENT**. The execution of a class name would patch the **ACTIVECLASS** variable to allow searching the appropriate class. At the end of the search order, the **ACTIVECLASS** vocabulary would be set null. This implementation should not conflict with any other special vocabulary constructs, such as **ONLY**.

Application and Implementation

The general use for classes is to organize the dictionary according to the types of objects being used. For example, one could use the phrases "**SINGLE +**" to add single-length integers, "**DOUBLE +**" to add double-length integers and "**FLOAT +**" to add floating-point numbers if the classes **SINGLE**, **DOUBLE** and **FLOAT** had been defined to describe single-length integers, double-length integers and floating-point numbers respectively. Figure One lists the code for the same sample application which is used in the Smalltalk book. I have used the **ONLY** concept to avoid the necessity of writing **SINGLE** before each of the single-length operations, and I have left the implementation of an integer dictionary class to the readers. The example uses three operations from the IntDictionary class: (1) **at** to access the value corresponding to a certain code; (2) **isAt** to store a value corresponding to a certain code; and (3) **new** to create a new IntDictionary given the maximum number of codes involved. The Forth code and the usage examples should be relatively straightforward. However, it may be useful to point out that the words corresponding to the dictionary codes for income and expense categories are not defined in the example; these definitions are not essential to understanding the example and are of the form

codeValue **CONSTANT** codeName

Figure Two lists the code for implementing the words I have proposed under the Laxen/Perry F83 model. The code should be relatively straightforward, so I will only review some of the more challenging sections. The **CLASS** defining word produces a dictionary entry similar to that of the **VOCABULARY** defining word with the addition of space for a pointer to the class's superclass and a different run-time action. **DEFER** compiles the code address of its run-time word **(DEFER)** and a counted string representation of the word which follows it in the input stream. **(DEFER)** extracts the address of the string which follows it, moves the instruction pointer past the string, looks up the word in the dictionary and either executes it or types an error message and aborts. **FIND** is modified by the addition of a call to **SEARCH-CLASS** before searching the **CONTEXT** and **CURRENT** vocabularies if the word

```
ONLY  FORTH  ALSO CLASS@ SINGLE

CLASS FinancialHistory (SUPER Object
CLASS@ FinancialHistory DEFINITIONS

: cashOnHand    (S 'hist -- 'n )            ;
: incomes       (S 'hist -- 'dict )  2+   @ ;
: expenditures (S 'hist -- 'dict )  4 +   @ ;

: initialBalance   (S 'dict1 'dict2 n -- )
  CREATE  ,  SWAP , , ;
: new               (S 'dict1 '-
  CREATE  0 ,  SWAP , , ;

: totalReceivedFrom   (S code hist -- n )
  incomes IntDictionary at ;
: totalSpentFor       (S code hist -- n )
  expenditures IntDictionary at ;

: receive   (S code n hist -- )
  2DUP cashOnHand +!
  SWAP  >R 2DUP totalReceivedFrom  R> +  SWAP
  incomes IntDictionary isAt ;
: spend     (S code n hist -- )
  OVER NEGATE OVER cashOnHand +!
  SWAP  >R 2DUP totalSpentFor  R> +  SWAP
  expenditures IntDictionary isAt ;

FORTH DEFINITIONS

Usage Examples

100 IntDictionary new HouseIncome
100 IntDictionary new HouseExpenses
ONLY  FORTH  ALSO CLASS@ FinancialHistory
HouseIncome HouseExpenses 350 initialBalance Household
utilities 32 Household spend
food 30 Household spend
rent 400 Household spend
wages 1000 Household receive
taxRefund 200 Household receive
Household cashOnHand @ .
```

Figure One
Example Application

is not found in the active class and by the addition of code to set the active class to null at the end of the search process. **SEARCHCLASS** simply follows the class's superclass chain while calling **(FIND)** to search each class's linked list of words along the way.

One possible concern in implementing this proposal is that it introduces another kind of prefix operator to the code-field prefix operators already proposed with multiple-code-field words. One might run into situations where a phrase of the form "CodePrefix ClassName WordName" must be handled. The implementor must ensure that the prefix operators act properly without interfering with each other. One would not want to try to execute the nonexistent second code field of ClassName, for instance. A simple solution would be to implement the code field prefix operators so that they check for intervening class prefix operators or so that the code-field prefix operator sets a system variable which is referred to in determining which code field of a multiple-code-field word to execute; there are many ways that this might be done. It seems logical to require that there be no intervening prefix operators between the class name and the word name.

```
VARIABLE ACTIVECLASS  ( pointer to class to be searched )
VARIABLE NEWCLASS     ( pointer to class being defined )
#THREADS 2* 2+ CONSTANT 'SUPER ( offset to superclass ptr. )

0 ACTIVECLASS !

Class Definition Words

: CLASS   (S -- )
  CREATE  IMMEDIATE  HERE NEWCLASS !
  #THREADS 0  DO  0 ,  LOOP
  HERE  VOC-LINK @ ,  VOC-LINK !  0 ,
  DOES>  ACTIVECLASS ! ;
: <SUPER   (S -- )
  ' >BODY  NEWCLASS @ 'SUPER +  ! ;
: CLASS@   (S -- )
  ' >BODY  CONTEXT ! ;

Class Variables

: CLASSVAR   (S -- )
  CREATE  0 ,
  DOES>  @  ACTIVECLASS ! ;
: CLASS>     (S -- )
  ' >BODY  ' >BODY  ! ;
: (DEFER)    (S -- )
  R> COUNT  2DUP + >R DROP  FIND IF EXECUTE
  ELSE  COUNT TYPE  TRUE ABORT" is undefined." THEN ;
: DEFER      (S -- )
  COMPILE (DEFER)  BL WORD  C@ 1+ ALLOT ; IMMEDIATE

Dictionary Lookup Modifications

: SEARCHCLASS   (S addr -- cfa flag ! addr false )
  FALSE  BEGIN
    DUP  ACTIVECLASS @  SWAP 0= OVER AND  WHILE
    DUP 'SUPER + @  ACTIVECLASS !
    SWAP DROP OVER SWAP  HASH @  (FIND)
  REPEAT ;
: FIND          (S addr -- cfa flag ! addr false )
  SEARCHCLASS  DUP 0= IF
    ( FIND as defined in F83 )
  THEN  0 ACTIVECLASS ! ;
```

Figure Two
Example Implementation

An Open Question

One of the most difficult questions to answer in the object-oriented programming model concerns the handling of generic classes of composite objects, such as arrays or stacks. How can one efficiently implement a generic array class where subclasses may be simply instantiated for byte arrays, bit arrays, double-length arrays or multi-dimensional arrays of these as they are needed? The solutions I have seen written in Smalltalk seem to be rather inefficient. Charles Moore did not include an **ARRAY** word in his initial design of Forth for basically this reason. I am considering several techniques, but perhaps someone out there already has a solution.

Conclusion

The principal benefit of the proposed approach is that it seems to solve the perceived problems without drastically complicating or changing the present character of Forth. Marriages of Forth and Smalltalk such as Kriya Systems's Neon provide more of Smalltalk's explicit structure at the expense of Forth's flexibility. I find that approach to be overly complex, although I should express my thanks to the implementors of Neon for provoking me to think about this subject. Ultimately, in the author's opinion, the responsibility for the production of elegant, clear and powerful software rests with the programmer. A language should provide a few simple yet powerful and carefully integrated constructs; the discipline and imagination of the programmer provide the rest.

Bibliography

1. Duff, Charles and Norman Iverson. "Forth Meets Smalltalk" in *Journal of Forth Application and Research*. Vol. 2, no. 3, pp. 7-26.
2. Goldberg, Adele and David Robson. *Smalltalk-80: The Language and its Implementation*. Reading, MA: Addison-Wesley Publishing Company, 1983.
3. Lyons, George. "Type Declarations" in *1980 FORML Proceedings*. pp. 72-74.
4. Moore, Charles. Interview on factorization in Leo Brodie. *Thinking Forth*. Englewood Cliffs, NJ: Prentice-Hall, Inc., 1984. pp. 196-197.
5. Laxen, Henry and Michael Perry. *Forth-83 Implementation Model.*
6. Perry, Michael. "Vocabulary Mechanisms in Forth" in *1980 FORML Proceedings*. pp. 39-41.
7. Ragsdale, William. "The ONLY Concept for Vocabularies" in *1982 FORML Proceedings*. pp. 109-116.
8. Rosen, Evan. "High Speed, Low Memory Consumption Structures" in *1982 FORML Proceedings*. pp. 191-196.

(Continued from page 18.)

```
SCR#   9
\ Pattern and fudge handlers                    NG 04/21/86
: FUDGE_MESSAGE   (   ---   )
   CR   ." Shall U&u (vee), Y&y (eye), and W&w (two vees) "
        ." be counted? (Y!N): "   ;

: COAX_#FUDGE   (   ---   )   \ prompt user
   BEGIN  FUDGE_MESSAGE KEY DUP EMIT DUP YES_OR_NO?
          IF    YES? #FUDGE !  TRUE   \ yes?, then fudge on
          ELSE  DROP  Y!N_MESSAGE  FALSE   \ invalid response
          THEN
   UNTIL   ;

: COAX_PATTERN   (   ---   )
   .PATTERN_MESSAGE  ( prompt )  READNAME  ( receive name )
   CR COAX_#FUDGE  ( also fudged num'ls )   ;

SCR#   11
\ Print analysis of name                        NG 04/18/86

: ANALYZE
\ Print formatted analysis of the pattern name
   CR CR  NAME SPAN? @ TYPE  ASCII : EMIT  CR CR
   #M PLOP  ." #Ms =" 1000 FLOP
   #D PLOP  ." #Ds ="  500 FLOP
   #C PLOP  ." #Cs ="  100 FLOP
   #L PLOP  ." #Ls ="   50 FLOP
   #X PLOP  ." #Xs ="   10 FLOP
   #V PLOP  ." #Vs ="    5 FLOP
   #I PLOP  ." #Is ="    1 FLOP
   10 SPACES 4 0 DO  ASCII - EMIT  LOOP  CR
   14 .R  CR   ;
```

```
SCR#   10
 0 \ Constant of the Beast                      NG 04/18/86
 1
 2 666 CONSTANT BEAST#   \ The Beast revealed!!!
 3
 4 : THE_BEAST?   ( n --- f )   \ True if the number is The Beast#
 5    BEAST# =   ;
 6
 7 : PROCLAIM   ( f ---   )   \ Announcing the discovery
 8    DARK BEEP CR CR CR  5 SPACES
 9    ." The number of " NAME SPAN? @ TYPE SPACE ." is 666," CR
10    ."      the number of The Beast of the Book of Revelation!"
11    12 0 DO  CR  LOOP   ;
12
13 : PLOP   ( addr --- n )   @ DUP 3 .R SPACE   ;
14
15 : FLOP   ( n ---   )   *  5 .R CR   ;

SCR#   12
 0 \ Word to call the Beast                     NG 04/21/86
 1
 2 : DONE?   (   --- f )   \ False means another try
 3    BEGIN  CR  ." Another name? (Y!N): "  KEY DUP EMIT
 4    DUP YES_OR_NO? NOT  WHILE  Y!N_MESSAGE  REPEAT
 5    CR  YES? NOT   ;
 6
 7 : BEAST?   \ Runs the analysis. User is prompted during LOADing.
 8    BEGIN
 9      INIT-#REGS
10      COAX_PATTERN  PATTERN-SCAN
11      NUMBER? DUP  THE_BEAST?
12      IF  DROP PROCLAIM  ELSE  ANALYZE  THEN
13      DONE?
14    UNTIL
15    CR CR CR CR ."      DONE!"   12 0 DO  CR  LOOP   ;
```

ATTENTION FORTH AUTHORS!

Author Recognition Program

To recognize and reward authors of Forth-related articles, the Forth Interest Group adopted the following Author Recognition Program, effective October 1, 1984.

Articles

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

a. The membership awarded is for the membership year following the one during which the article was published.

b. Only one membership per person is awarded in any year, regardless of the number of articles the person published in that year.

c. The article's length must be one page or more in the magazine in which it was published.

d. The author must submit the printed article (photocopies are accepted) to the Forth Interest Group, including identification of the magazine and issue in which it appeared, within sixty days of publication. In return, the author will be sent a coupon good for the following year's membership.

e. If the original article was published in a language other than English, the article must be accompanied by an English translation.

f. Articles are eligible under this program only if they were first published after October 1, 1984.

Letters to the Editor

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

a. The credit applies only to membership fees for the membership year following the one in which the letter was published.

b. The maximum award in any year to any person will not exceed the full cost of the membership fee for the following year.

c. The author must submit to the Forth Interest Group a photocopy of the printed letter, including identification of the magazine and issue in which it appeared, within sixty days of publication. The author will then be sent a coupon worth $10 toward the following year's membership.

d. If the original letter was published in a language other than English, the letter must be accompanied by an English translation.

e. Letters are eligible under this program only if they were first published after October 1, 1984.

Ultimate CASE Statement

Wil Baden
Costa Mesa, California

Many citizens of the Forth community have lamented the lack of a **CASE** statement in standard Forth language specifications. Since the first rule of Forth programming is, "If you don't like it, change it," there have been many proposals, and *Forth Dimensions* even held The Great CASE Contest in Volume II. Although the winning entry of that contest, submitted by Charles Eaker, has been widely implemented and even offered as part of many vendors' systems, the flood of proposals has not ceased. There have been many articles and letters on the subject in *Forth Dimensions.*

All proposals to date have had problems. Portability is one. Another is that they all have been too specialized and restricted in their area of application. Generalization is accomplished by designing another special case of **CASE**.

Strictly speaking, a **CASE** statement is unnecessary. It is "syntactic sugar" to make a program easier to write, read and understand. It is so helpful in doing this that it is a standard feature of all other modern programming languages.

Figure One-a is a rather futile program written in C to illustrate a common pattern of logical decisions in many programs. ("= =" is "equal to" for comparing two things, to distinguish it from "=" for assignment as in Fortran or Basic.) An equivalent Forth version would look something like Figure One-b.

Most people will agree that Figure One-a would be better written as in Figure Two-a. An even better way is found in some dialects of C, illustrated by Figure Two-b. In this extension, following syntax from Pascal, values separated by "," indicate a set of values, and values separated by ".." indicate a range.

Some Forth proposals have one definition for individual values and another definition for a range of values. There would have to be another definition for a set of values. No earlier Forth proposal that I know of allows sets and ranges together, as in:

case 2..3, 12:

What is proposed here is a single **CASE** statement for Forth which will include all these variations, and many more, that can be implemented in fig-FORTH, Forth-79, Forth-83 and any other Forth.

Figure Two-a would look as shown in Figure Three. Let's add two more spoons of syntactic sugar, as in Figure Four. As has been noted elsewhere, too much syntactic sugar causes semantic diabetes. Our **CASE** is sweet enough. Figure Five is an example to show some of the possibilities.

Now for a real life example. Figure Six is a recension of a word in John James' "Universal Text File Reader" (*Forth Dimensions* VII/3). One of my favorite examples is "Thirty days hath September, April, June and November ...." See Figure Seven.

If **NUMBER** in your system is vectored, you may want to replace it in some

```
craps(n)
int n;
{  if (n == 7)
      printf("You win");
   else if (n == 11)
      printf("You win");
   else if (n == 2)
      printf("You lose");
   else if (n == 3)
      printf("You lose");
   else if (n == 12)
      printf("You lose");
   else printf("%d is your point",n);
}
```

Figure One-a

```
: CRAPS ( n -- )
   DUP 7 =
   IF DROP ." You win"
   ELSE DUP 11 =
      IF DROP ." You win"
      ELSE DUP 2 =
         IF DROP ." You lose"
         ELSE DUP 3 =
            IF DROP ." You lose"
            ELSE DUP 12 =
               IF DROP ." You win"
               ELSE . ." is your point" THEN
   THEN THEN THEN THEN THEN ;
```

Figure One-b

```
craps(n)
int (n);
{  switch(n) {
   case  7:  printf("You win"); break;
   case 11:  printf("You win"); break;
   case  2:  printf("You lose"); break;
   case  3:  printf("You lose"); break;
   case 12:  printf("You lose"); break;
   default:  printf("%d is your point",n);
   }
}
```

Figure Two-a

```
craps(n)
int n;
{  switch(n) {
   case 7, 11:    printf("You win"); break;
   case 2..3, 12: printf("You lose"); break;
   default:       printf("%d is your point",n);
   }
}
```

Figure Two-b

```
: CASE   DUP ;

: CRAPS  ( n -- )
   CASE   7 = IF DROP ." You win" EXIT THEN
   CASE  11 = IF DROP ." You win" EXIT THEN
   CASE   2 = IF DROP ." You lose" EXIT THEN
   CASE   3 = IF DROP ." You lose" EXIT THEN
   CASE  12 = IF DROP ." You lose" EXIT THEN
   . ." is your point" ;
```

Figure Three

```
: OF   ( n flag -- ) [COMPILE] IF   COMPILE DROP ; IMMEDIATE
: =OR  ( n flag n -- n flag ) 2 PICK = OR ;

: CRAPS  ( n -- )
   CASE  7 =  11 =OR          OF ." You win" EXIT THEN
   CASE  2 3 BETWEEN  12 =OR  OF ." You lose" EXIT THEN
   . ." is your point" ;
```

Figure Four

```
: WHATEVER  ( n --)
   CASE  0=                          OF ." Zero" EXIT THEN
   CASE  0<                          OF ." Negative" EXIT THEN
   CASE  DUP 1- AND 0=               OF ." Power of 2" EXIT THEN
   CASE  ASCII 0  ASCII 9  BETWEEN OF ." Digit" EXIT THEN
   CASE  ASCII ,  ASCII /  BETWEEN
         ASCII : =OR          OF ." Punctuation ,-./: " EXIT THEN
   DROP  ." Whatever" ;
```

Figure Five

```
: ?OUT  ( c -- )  127 AND
   CASE  0=  13 ( return) =OR
     OF  ?NEW-LINE  EXIT THEN
   CASE  10 ( linefeed) =  12 ( formfeed) =OR
     OF  #BLANK-LINES @ 0=
         IF  ?NEW-LINE  THEN
         EXIT THEN
   0 #BLANK-LINES !
   CASE  32 <
     OF  ( Do nothing.)  EXIT THEN
   EMIT ;
```

Figure Six

```
: LEAPYEAR?  ( -- tf : true when the year is a leap year.)
 #YEAR @
   CASE 400 MOD 0= OF  TRUE  EXIT THEN
   CASE 100 MOD 0= OF  FALSE EXIT THEN
   CASE   4 MOD 0= OF  TRUE  EXIT THEN
   DROP  FALSE ;

: DAYS  ( month# -- days-in-month )
   CASE  9 =  4 =OR  6 =OR  11 =OR  OF  30  EXIT THEN
   CASE  2 = NOT                    OF  31  EXIT THEN
   DROP  LEAPYEAR? IF  29  ELSE  28  THEN ;
```

Figure Seven

```
: CBASE!  ( a c -- a' )
   CASE  ASCII $ =  OF  HEX     1+ EXIT THEN
   CASE  ASCII @ =  OF  OCTAL   1+ EXIT THEN
   CASE  ASCII % =  OF  BINARY  1+ EXIT THEN
   CASE  ASCII & =  OF  DECIMAL 1+ EXIT THEN
   DROP ;

: BASE-NUMBER  ( a -- d )
   BASE @ >R   DUP 1+ C@ CBASE!
   NUMBER?   R> BASE !   0= ABORT" ?" ;
```

Figure Eight

```
HEX
: CLASSIFY  ( n -- )
   CASE  20 <  7F =OR      OF ." Control character" EXIT THEN
   CASE  20 2F BETWEEN
   OVER  3A 40 BETWEEN OR
   OVER  5B 60 BETWEEN OR
   OVER  7B 7E BETWEEN OR  OF ." Punctuation"       EXIT THEN
   CASE  30 39 BETWEEN     OF ." Digit"             EXIT THEN
   CASE  41 5A BETWEEN     OF ." Upper case letter" EXIT THEN
   CASE  61 7A BETWEEN     OF ." Lower case letter" EXIT THEN
   DROP                       ." Not a character" ;
```

Figure Nine

```
CREATE CASE   ' DUP ( CFA ) @ ' CASE ( CFA ) !
```

Figure Ten-a

```
: =OR   ( n tf n -- n tf ) 3 PICK = OR ;
```

Figure Ten-b

```
: =OR  ( n tf n -- n tf ) >R  OVER R> = OR ;
```

Figure Ten-c

```
: WITHIN   ( n n1 n2 -- tf : true when n1 <= n & n < n2.)
   OVER - >R   -  R> U< ;
: BETWEEN  ( n n1 n2 -- tf : true when n1 <= n & n <= n2.)
   WITHIN 1+ ;

: ASCII  ( ^ c -- c : integer value of character c.)
   BL WORD COUNT 1- ABORT" ?"   C@ STATE @
   IF  [COMPILE] LITERAL  THEN ; IMMEDIATE
```

Figure Eleven-a

```
: HEX  ( -- ) 16 BASE ! ;
: OCTAL  ( -- ) 8 BASE ! ;
: BINARY  ( -- ) 2 BASE ! ;
: DECIMAL  ( -- ) 10 BASE ! ;

: NUMBER?  ( addr -- dn tf ) 0 0 ROT CONVERT C@ BL = ;
```

Figure Eleven-b

applications with a version that selects the numerical radix according to the first character. Figure Eight implements a convention used on Motorola systems (e.g., 68000). Laxen's **CLASSIFY** example (*FD* VII/1) can be written without redundant classes with no additional definitions, as in Figure Nine.

Since **DUP** is assembler code, in most systems you can optimize its definition with something like that in Figure Ten-a. The Forth-79 definition of **=OR** is given in Figure Ten-b. If you do not have **PICK**, as in fig-FORTH, or if **PICK** is not an assembler code definition, see Figure Ten-c.

A **CASE** statement in any programming language is intended for a series of tests to classify a value. To do this in other languages without using a **CASE** structure would require repeating the value at each test, giving a tedious appearance to the source. In Forth, the data stack allows us to avoid such explicit references to the value. In Forth, a **CASE** statement has the pattern **DUP** ... **IF DROP** .... We have sweetened this to **CASE** ... **OF** ....

The trivial nature of the implementation emphasizes that a **CASE** statement is not essential to Forth. Those Forth practitioners who pride themselves on how lean and mean their Forth is will find it superfluous. My intent is not to propose this definition of **CASE** for standardization; but on the other hand, any further **CASE** proposal should be as simple to implement, as portable and as powerful.

Auxiliary Definitions

You may already have some of these. Your definitions may be different from those shown in Figure Eleven-a. **#BLANK-LINES** and **?NEW-LINE** are words peculiar to the application. **#BLANK-LINES** is a variable counting the number of successive blank lines. **?NEW-LINE** does a CR when the value of **#BLANK-LINES** is less than two.

Figure Eleven-b provides definitions for several fundamental Forth words. It also presents a naive version of **NUMBER?** that ignores details such as sign and punctuation, and is not intended for actual use.

Volume Seven Index

This reference guide to Volume VII was prepared as a service to our readers. Items are referenced by issue number and page number; the first entry refers to an article in volume VII, issue 1, page 36.

'86 National Forth Convention

Nearly one thousand people gathered in November to explore the state of "Forth Engines." Crowds in the exhibition area were larger and more animated than at previous years' events, showing great interest in the research and large commercial ventures based on Forth software and hardware. The annual event was held at the new Santa Clara Trade and Convention Center in California's Silicon Valley. The spacious facility easily accommodated the large lecture hall, exhibition hall and three separate meeting rooms, where concurrent sessions were held for the two days.

Speakers explained several proven approaches to embedding Forth in hardware. Novix's NC4000 and NC6000 chips, and products incorporating them, were of the expected interest to attendees, as was the Hartronix engine's use as a robotics controller. Other systems discussed by featured speakers were Zilog's Super Z-8 and Rockwell's R65F11 and F68HC11 chips. New to most attendees were the thirty-two-bit Forth chip developed by Johns Hopkins University and the multi-stack, writeable instruction set computer (WISC) from Haydon Enterprises. The spectrum of design approaches was well represented; it is to be hoped that a well-written set of Forth benchmarks will appear in order to efficiently compare the relative strengths of each.

Future of Forth Engines

The last speakers' session was dedicated to a panel that discussed forseeable trends in this field. The panel consisted of experts who have done extensive work in the theory, design and development of Forth engines. Chaired by Martin Tracy of Forth, Inc., the panelists were Gary Feierbach (Inner Access), Glen Haydon (Haydon Enterprises), Charles Moore (Computer Cowboys) and John Rible (Novix, Inc.). Questions were taken from the audience.

What would you like to see in terms of recognition of Forth?

Charles Moore stated that he would like to see Forth on the list of government-approved languages. Gary Feierbach would like Forth to be recognized across a broad spectrum of application areas. The relocatable library question should be addressed satisfactorily. That some Forth systems permit compilation at the same rate as linking in other languages should be a factor in gaining recognition — a complete investigation would be persuasive, but initial exposure to a less-than-optimal Forth system can slow acceptance.

Glen Haydon then pointed out that Phil Koopman has a Forth library system available through Mountain View Press. Regarding the merits of advertising, the best approach to getting something across is having a job well done and well received, which addresses and solves the problem at hand. When we show that, Forth predominates. Charles Moore responded by saying that advertising convinces users we are a serious entry in the marketplace, so we must maintain a public relations image.

Where are the optimizing compilers that will make the Forth engine more widely useful?

John Rible said Small-C is available for the Novix 4000. Others are under negotiation and they are expensive. But why do it six to seven times faster (than an IBM AT) in C on top of Forth hardware, when it could be forty times faster in native Forth?

What will the second generation of Forth engines look like?

Charles Moore stated that any engine one wants is producible. One consequence of the simplicity of the Forth processor is that it can be easily combined with other hardware (on-chip stacks, multi-processors in a single chip, etc.). He doesn't think future generations will have the same thousand-fold increases or the same impact.

John Rible added that the behavior of the processor is dependent on the rest of the world. They are doing what they can with the current technology. Hopefully, the computer theorists will learn that one or two stacks will speed things up dramatically. Gary Feierbach expects us to see thirty-two-bit chips and custom chips for specific applications. He also believes we need a targeted education effort so prospective users can see what can be done with Forth in hardware.

Glen Haydon concluded that five years from now we will still have eight-bit processors, and the sixty-four-bit processors will be where the thirty-two-bit ones are at today. Whatever happens, keeping it simple will keep it on track with Forth theory. The bottleneck today is still memory speed. The cost of memory will continue to lower, and speed will increase. Designs for Forth engines will change according to what there will be time to do between memory accesses.

How can Novix address a customer's need for a bugless engine, a full implementation of the chip as it was originally intended and reliable delivery?

John Rible related that Novix has licensed some rights to Harris Semiconductor, and that they are working with it in their core cell library. Novix is upgrading to the NC6000 and is committed to fixing the NC4000. The rest is up to the marketplace and to management.

Charles Moore compared the situation to the chicken-and-egg syndrome: if anyone had ordered 10,000 chips, it would have been different. It is clearly not desirable to order a chip with bugs or which may not be readily available. Novix is trying its best in a field dominated by giants.

The pinout is very large for these processors, keeping them expensive. What about Forth chips with fewer pinouts?

Charles Moore said he could visualize a twenty-four pin, eight-bit processor, but couldn't see anything useful smaller than that. It is a manufacturing and quantity problem, not so much one of design. Pins are cheap in terms of cost/benefit tradeoffs, especially considering the finding in neural net research that a high degree of interconnectivity can yield interesting results.

How do you see casting Forth into hardware engines as changing the Forth language?

Charles Moore: By keeping the program memory small but giving lots of space to the stack.

John Rible: They are providing improved addressing space, but it won't be terribly useful except in stacks. Using stack pointers into larger areas of memory becomes interesting. There isn't enough experience at programming these chips to know what kind of operations can usefully occur in an overlapped manner. Someday we will be able to write truly portable code that can be compiled into these processors to give us the full power of that processor without the programmer having to serve as the compiler.

Glen Haydon: Chuck outlines the forty-five or so necessary functions for Forth as it stands. That should be fairly solid. In the future we will look at what other functions will be simple and necessary, and whether they can be combined efficiently with other operations. The basic Forth kernel may grow by twenty or so words.

Concurrent Sessions

Well-known Forth experts conducted tutorials on subjects such as multitasking, target compilation, vectored I/O and control structure extensions. Groups of users met with the vendors of Mach 1 and Mach 2 (68000 systems), polyFORTH, MVP-FORTH, MacForth and MultiForth, F83, and the NC4000. Special seminars discussed managing Forth programmers and writing Forth-related articles. There was a report from the 1986 FORML journey to present technical papers in China; a meeting of FIG Chapters representatives; a FIGGRAPH caucus about Forth's use in current graphics technology; and the annual "fireside chat" with Mr. Charles Moore, original developer of the Forth language.

National FIG Meeting

This year's convention saw a special meeting for FIG members, chaired by President Robert Reiling. Other Board members, all present at this meeting, include Martin Tracy, Vice-President; Kim Harris, Secretary; John Hall, Treasurer; and Thea Martin. Mr. Reiling described the Forth Interest Group as a non-profit organization that is tax exempt, reporting to the State of California and to the U.S. Internal Revenue Service. It has about 4000 members, one quarter of whom live outside the United States. FIG services and activities are supported by members' dues, by a modest income from the sale of publications and by advertisers in *Forth Dimensions*. The Forth National Convention itself has been managed so that income and expenses are about equal.

Early last year, a small group of board members and other key figures met at their own expense in a two-day, think-tank style retreat. They addressed issues such as membership, services, growth and how FIG's position addresses the general software/languages community. A good deal of information was solicited in advance from a cross-section of members and Forth vendors, and aided greatly in all the discussions. This event, and any similar meetings that may follow, will serve to focus attention on key issues of concern and benefit to the entire community.

FIG's growth mandated this kind of intensive session for planning and definition of important directions.

Some results of this initial retreat were the FIG Model Library developed by Martin Tracy, health and life insurance options for members, the mechanism for adding or deleting publications from FIG's ordering list, streamlined financial operations (including improved monthly reporting on budget, P&L and inventory) and changes in FIG's by-laws.

Revision of the FIG by-laws is of particular note among recent actions taken by the Board of Directors. Board member Thea Martin saw defficiencies in the provisions regarding members' responsibilities. Only five people had started FIG, and only the Board was officially imbued then with both responsibility and authority to act on FIG's behalf. It was a close-knit and efficient way of conducting business.

After thorough review, the Board has formally amended the by-laws. The essential change now directs a Nominating Committee to report to the entire FIG membership (probably in *Forth Dimensions*). The committee can accept nominations for board member candidates from the membership at large. Names must be submitted to the committee along with the supporting signatures of ten FIG members. The committee will notify the membership of nominees' names, election dates and a vote-by-proxy mechanism.

FIG's normal business activities are directed by a volunteer business group that meets monthly in San Jose, California, with several Board members normally in attendance along with other professional associates and interested members. Day-to-day operations are carried out by the Association Development Center (Shepherd Associates), a paid service with whom FIG works closely.

FIG Chapters exist in many parts of the world. At the time of this meeting, there were eighty-seven active chapters, with others in various stages of formation. In many ways, they are the volunteer-based foundation of the organization. On the 1986 FORML tour that visited China, Forth experts there exhibited great interest in forming a FIG Chapter. Such a chapter would be the first on the mainland and would serve a great number of Forth users. Like a number of countries, however, certain currency regulations make it difficult to get the five FIG members necessary to form an official FIG Chapter. As a result, Shanghai's prestigious Jiao Tong University was made an Associate FIG Chapter for a period of two years. Welcome!

The keynote speaker of the concluding FIG banquet was John Peers, President and CEO of Novix, Inc. His amusing style, strong convictions and philosophy, combined with his extensive high-tech background, made Mr. Peers an informative and entertaining guest. Also at this banquet, Dr. C.H. Ting was announced as the recipient of the "Figgy" award, for volunteer activities that have done much to advance the cause of Forth during the past year. In addition to work that includes several popular books on the FIG Order Form, Dr. Ting was the Program Chairman for this year's convention. A good job, well done!

—Marlin Ouverson

Index to Advertisers

ATTENTION FIG MEMBERS! WE NEED YOUR HELP

At the FORTH Interest Group we know Forth is being used in many sophisticated and complicated projects. Unfortunately, the Forth community has never compiled a complete reference document summarizing how and where Forth is being used. We believe this type of document would be very helpful to both the novice considering learning Forth and the professional experiencing corporate resistance to using it.

Would you please help us put one together? All you need to do is complete the questionaire below and return it directly to us by March 15! All completed questionaires should be mailed to: Forth Interest Group, P.O. Box 8231, San Jose, CA 95155.

1. Company name and address: _____

   _____

   _____

2. Name of the programmer _____
   (Note: for internal use only. Will not be published.).

3. Project or product name _____

4. Date project or product completed _____

5. Was the project: For sale to an end user? _____ yes _____ no
   For in-house use? _____ yes _____ no
   For OEMs? _____ yes _____ no

6. Indicate approximate number of users:
   _____1-50 _____301-400
   _____50-100 _____401-600
   _____100-200 _____?
   _____200-300

7. Is Forth hidden from the user? _____ yes _____ no

8. Briefly describe the project (30 words) _____

   _____

   _____

9. Briefly describe the benefits of using this project or product. _____

   _____

   _____

Thank you for your participation. If you would like a copy of the results please complete the following.

Name _____

Company _____

Address _____

_____

City, State, Zip _____

FIG Chapters

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

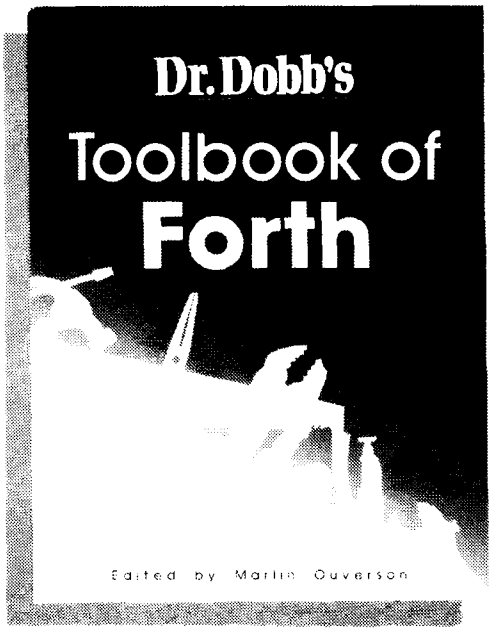
Apple Corps Forth Users Chapter
Twice Monthly, 1st &
3rd Tues., 7:30 p.m.
1515 Sloat Boulevard, #2
San Francisco, CA
Call Robert Dudley Ackerman
415/626-6295

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

NOW AVAILABLE



Dr. Dobb's Toolbox of Forth is a comprehensive collection of useful Forth programs and tutorials that contain expanded and revised versions of DDJ's best Forth articles along with new Forth material.

You'll also find appendices that will help you convert fig-Forth to Forth-83, and tell you how to stay up-to-date on the latest developments of Forth.

FORTH INTEREST GROUP fig

$23 EACH

FORTH INTEREST GROUP fig

FROM THE FORTH INTEREST GROUP

FORTH INTEREST GROUP
P. O. Box 8231
San Jose, CA 95155