$10 Volume XIII, Number 3 September 1991 October

FORTH DIMENSIONS

Neural Network Tools (II)

Universal Control Structures

Ada Multiprocessor Real-Time Kernel

Contents

Features

Departments

Editorial

A last reminder: *FD* has announced a call for papers about object-oriented programming. The closing date is September 16 for prize consideration—see our ad on page 37 for prize information, and refer to the editorial in issue XIII/1 for other details that may be helpful when preparing your paper. (Papers received after the deadline will still be evaluated for publication.)

So, Why Don't You Write?

FD is also looking for many new articles from its readers to fill the editorial coffers. This is, after all, a reader-written publication—this means you! We are looking for a wide range of topics, as eclectic as our readership. Chances are, if you find something interesting, clever, or challenging, many others will, too. If you aren't sure, just drop me a note that outlines the idea, and I will try to give meaningful feedback to you. (And check out our "Author Recognition Program," details of which are reprinted on page 35.)

If you aren't sure if your idea is really a new one, or can't remember whose original work it derives from—to say nothing of the times when you wonder whether the blank wall you've been staring at has been scaled already by some other Forth explorer—there are some FIG reference tools you might want to get. Look on the FIG Mail Order Form for the "We're Sure You Wanted to Know..." series. The Forth Bibliography from Rochester also provides Forth reference information from outside the FIG domain, but hasn't been updated since January 1987. Get a copy anyway for an inspirational look at how much Forth material really has been circulating through the years.

Chinese National Exam of Forth Programmers

China has officially announced a national examination to be held in September. Great importance is often attached to the results of such examinations. As Dr. Ting relates, the Chinese love examinations, a tradition of 2000 years. For your interest, following is a translation of part of the announcement:

To accelerate the application of Forth in various technical fields, to enlarge the ranks of Forth programmers, to raise the expertise in the Forth language, and to satisfy the increasing need for Forth users, the Chinese Scientific Instrumentation Society, the Chinese Software Association, and the Chinese Forth Interest Group will jointly sponsor a National Examination of Forth Programmers on September 8, 1991 in various provinces and cities across China. The sponsors will grant certificates to programmers passing this examination. They will also select the best ten Forth programmers and thirty excellent Forth programmers, based on the examination, to receive special certificates and awards, to be recommended to participate in Forth projects, and to carry out advanced studies abroad.

Subjects of Examination

1. Computer Fundamentals (20%)
   Number systems and conversions
   Internal representations of numbers
   Major components in a computer and their relationships
   Instructions and instruction sets
   Characteristics and functions of registers
   Characteristics and properties of I/O devices
   Data structures (stacks and lists)
   Operating system fundamentals
   Software engineering fundamentals

2. Elementary Knowledge of

(Continued on page 34.)

Forth Dimensions
Volume XIII, Number 3
September 1991 October

Published by the
Forth Interest Group

Editor
Marlin Ouverson

Circulation/Order Desk
Anna Brereton

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

Subscription to *Forth Dimensions* is included with membership in the Forth Interest Group at $40 per year ($52 overseas air). For membership, change of address, and to submit items for publication, the address is: Forth Interest Group, P.O. Box 8231, San Jose, California 95155. Administrative offices: 408-277-0668. Fax: 408-286-8988. Advertising sales: 805-946-2272.

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

"*Forth Dimensions* (ISSN 0884-0822) is published bimonthly for $40/46/52 per year by the Forth Interest Group, 1330 S. Bascom Ave., Suite D, San Jose, CA 95128. Second-class postage paid at San Jose, CA. POSTMASTER: Send address changes to *Forth Dimensions*, P.O. Box 8231, San Jose, CA 95155."

Letters

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

Black-Belt Exhaustion & Lean, Mean FIG

Dear Phil Koopman, Jr.:

I was sorry to hear that you decided to drop your FIG membership *[see* FD *XIII/2]*. You are one of the very visible and highly acclaimed Forth programmers. It is a great loss to FIG, not having you continue your membership.

You correctly pointed out that FIG is having financial troubles. The main problem I see is that we have been losing members since 1984, when the membership peaked at about 4,500. The current membership of about 2000 cannot support the operation of FIG using a professional support organization like ADC.

The membership loss has puzzled me all these years. There are more computers, more users, but why is FIG getting leaner and leaner? We have tried many different things to attract more new members, as well as to keep the old ones, but nothing seemed to work. There were substantial improvements on the appearance and substance in *FD*. We put lots of effort into getting the GEnie Forth RoundTable going. We had gone out of our way to promote Forth to the outside, but to no avail. Harris put in great amounts of money and energy to promote Forth, and even assembled a very high-powered team of Forth black-belts, including yourself, to develop the turbo-charged Forth engines. Why did Harris abandon its efforts? I really hope that somebody will tell me what's going on and what we have to do to change the public perception of the inadequacies of Forth and FIG.

FIG has always been an organization of volunteers. The founders of FIG expended great energy to craft fig-Forth, which propelled FIG into a viable association. Other members like Mike Perry, Henry Laxen, Bob Smith, and Tom Zimmer improved on fig-Forth and contributed F83 and F-PC, which have kept FIG going. I am fully aware of your contributions to Forth, in MVP-FORTH and the WISC technology. However, may I ask what you have contributed to FIG as a member? You complained that there is not a FIG chapter in your area. Every chapter is started by somebody; have you considered starting one yourself? You certainly have the capability, the energy, and the charisma to lead a large group and to teach lots of newcomers. Maybe the reason that FIG is shrinking is due to the exhaustion of leaders like you?

I had not served on the FIG Board of Directors until last year. However, I have regularly attended the business meetings and watched FIG's operations for ten years. One thing I can say for the Board is that every dollar is accounted for and the records are kept accurately. All expenses are openly discussed and must be approved at these meetings. The books are available at the FIG office for members to inspect. *Forth Dimensions* will also be publishing financial reports. *[See the President's Letter in this issue.—Ed.]*

I appreciate that you took the trouble to write and tell us what you think of FIG. I fully understand your frustration, and respect your decision not to continue your membership. However, I think the greatest challenge we Forth programmers all face is not just to survive the onslaught of C and Unix, but to pool our resources together and find the niche where Forth can still grow and shine.

Let me repeat an ancient Chinese story to make my point. The King of Tsu met an old man, crying in the countryside. He asked the old man why he was crying. The old man said, "I presented this piece of rock to your grandfather because it contains a large jade. Your grandfather let his jade expert inspect it, and he said, 'It's just a stone.' So, your grandfather cut off my right leg. Then I presented it to your father. Your father had it inspected, and he cut off my left leg. I cannot walk any more. Then Heaven brought you here so I can see you. Here is the rock. It's yours, if you can use it." The King of Tsu ordered his jade worker to saw it open and found the largest and the most beautiful jade ever seen in China. It was from this jade that the First Emperor of China made his imperial seal.

Forth is like this piece of jade in a rock. We know it is the best tool for human beings to control computers. We tried to give it to the world, but what did we get back? Both feet cut off. What can we do? I think all we can do is stick together, keep warm, and keep the pot stirred. If Forth is really that good, it will shine eventually. When? I do not think anybody knows. FIG is the only place where we can keep ourselves warm.

Dr. C.H. Ting
San Mateo, California

There's no reason why metacompilers should be hard to learn, use, or understand...

Mastering Metacompilation

Dear Editor,

Metacompilation is getting a bum rap.

Recent postings on GEnie, and an item by Dr. C.H. Ting in *Forth Dimensions* ("How Metacompilation Stops the Growth Rate of Forth Programmers," *FD* XIII/1), have re-opened one of the dark closets of Forth: metacompilation.

Metacompilation is probably the most arcane aspect of Forth, and is usually considered the province of the Forth "priesthood." I sympathize. But I think it's a mistake to abandon metacompilation for assembly language source, as Dr. Ting advocates. This is like going back to gas lighting because people get shocked by (and don't understand) electricity.

Education is the answer.

For the last four years, I've been saying that the problem is *not* that there aren't good Forth metacompilers, but that there isn't any good *documentation* for metacompilers.

I've come up through this the hard way. My first experience with a metacompiler (cross-compiler) was in 1982, when I tried to use a commercial product. The documentation was incomprehensible, the compiler obscure. Finally, I was able to complete my project by hacking my code onto the kernel supplied with the compiler, and by abandoning any attempt to use defining words, `IMMEDIATE` words, or vocabularies. (Rather like programming in C, come to think of it.)

My second experience was with a version of Cassady's Metaforth, implemented and modified by a friend of mine. I began to

Figure One. The probability of matching birthdays.

```
BIGSIZEMAX     size (an even number of bytes) of biggest number needed.

BIG            area of memory which holds a multiple-length unsigned
number, two bytes longer than BIGSIZEMAX since the most significant
cell is used to monitor overflow.

BIGLOW         address of least significant cell of BIG number.

BIG! ( u -- ) sets BIG number to u.

BIG@ ( -- u ) fetches u from least significant cell of BIG number.

BIGSIZE ( -- u ) returns the size (rounded up to an even number of
bytes) of BIG number, excluding leading zeroes.

BIG* ( u -- ) multiplies BIG number by u; product is new BIG number.

BIG/MOD ( u1 -- u2 ) divides BIG number by u1; quotient is new BIG
number, remainder is u2.

PEOPLE ( n -- ) calculates and displays the probability of at least one
pair of matching birthdays in a group of n people.
```

```
\ Big Number Arithmetic & The Probability of Matching Birthdays

100 constant BIGSIZEMAX  ( whatever you need but must be even )
variable BIG bigsizemax allot  big bigsizemax + constant BIGLOW
: BIG!     ( u -- ) big bigsizemax erase biglow ! ;
: BIG@     ( -- u ) biglow @ ;
: BIGSIZE  ( -- u ) big 2+ bigsizemax 0 skip nip 1+ -2 and ;
: BIG*     ( u -- ) 0 biglow bigsize - biglow
  do over i @ um* rot 0 d+ swap i ! -2 +loop  ( Forth-83 version )
  2drop big @ abort" *** big multiply overflow ***" ;
: BIG/MOD ( u1 -- u2 ) 0 biglow 2+ dup bigsize -
  ?do over i @ -rot um/mod i ! 2 +loop nip ;

: PEOPLE ( n -- ) >r 10000 dup big! 366 dup r@ - ?do i big* loop
  0 r> 0 ?do drop 365 big/mod loop big@ swap 182 > if 1+ then - 0
  ."   probability of a match is " <# # # # # 46 hold # #> type ;
```

understand the workings of the compiler, but even with my friend as a resource, I couldn't decipher how to add defining words or vocabularies.

The breakthrough came on my third attempt: the polyFORTH target compiler. And the reason for the breakthrough is not that the compiler is that much simpler (although it's certainly elegant!), but that I was able to attend the polyFORTH advanced course and have someone *teach* me how metacompilers work. Enlightenment!

I then proceeded to test my understanding of the theory and practice of metacompilers, by writing one of my own. (I had other reasons for this—I needed features that no one else was offering.) But, remembering how much difficulty I had in acquiring the essential concepts, I resolved to keep notes and record my thoughts *as I wrote the compiler.* I think I may be the first person to document the process of discovery this way.

In December of 1988 I condensed these notes into a one-hour presentation for our local FIG chapter, and was amply rewarded. Even though I delivered it at warp speed, several people came up to me afterward and told me that they understood metacompilation for the first time.

The lesson I learned from giving this talk is this: there's no reason why metacompilers should be hard to learn, use, or understand. The concepts aren't complicated; it's just that everyone who understands metacompilers takes them for granted!

Frank Sergeant, in *Forth Dimensions* XII/6, has started to dispel the mystery of metacompilation, by presenting a clear description of an elegant compiler. Like eForth, Pygmy is being ported to many CPUs in the embed-

(Continued on page 15.)

Universal Control Structures

Kevin Haddock
Chico, California

When I first started working with Forth, it's execution control was one of the most difficult aspects to learn. The basic words weren't difficult to understand—but I was encountering frustration trying to make my programs fit into the mold those words wanted to impose on them.

After seeing the wisdom of everything else Forth has to offer—simplicity, factoring, bottom-up design, performance, and economy, to name a few—I see evidence that control structures are an area of concern even for the enlightened few.

Indications of this can be found in the numerous efforts at case statements: Parnas' `IT ... TI` (*FD* VI/1), and security-less (e.g., poly-) Forth's gymnastic mixing and matching of incompatible types of control operators by bouncing offset addresses around on the stack at compile time.

I have always claimed that Forth has all the advantages of an interpreted language like BASIC, but here again the control structures get in the way. In BASIC, you can always whip up a quick loop or check the logic of your code with a conditional typed at the command line.

I have seen this interpretative situation addressed to some degree, but to date I have not seen an integrated solution that is uniform, clean, simple, and "Forth-like." (If I can judge what is "Forth-like"; the debate rages on!).

I was discussing the situation with a Forth-wiser friend of mine when I made the first step in stumbling upon the solution. Sometimes the hardest solution to see is the one that is right in front of your nose. (This happens a lot with me, especially when it comes to Forth.)

He said, "You know, Kevin, there really are only two control operators in Forth, `BRANCH` and `0BRANCH` (and `?BRANCH` in the newer versions). You can do whatever you want with those." Well, I knew that on an intellectual level, but this forced me to focus, it got my wheels turning. I know the wheels of my mind grind exceedingly slow, but I would like to think they grind exceedingly fine!

One of the things that created the most difficulty working with traditional Forth, in comparison with C, for example, was crafting the code to eliminate unnecessary tests. If any one of a series of ANDed conditions is not true, there is no need to waste any more time extracting and evaluating subsequent conditions. Similarly, if any one in a series of ORed conditions is true, processing can continue in the body of the condition. Since real-time is one of Forths' strengths, how come it falls so flat in this area? Control structures!

> ***Since real-time is one of Forth's strengths, how come it falls so flat in this area? Control structures!***

While writing my on-line classified ad system in Bourne Shell, I recall having fun seeing how many of my conditional needs could be met with the `||` (OR) and `&&` (AND) operators, which would drop out of the current line of script if the previous command returned true or false, respectively. I was pleasantly surprised at how comfortable it was working with this arrangement, although it required a slightly different mode of thought than is conventionally used in programming. I could appease my old modes of thought by coining the pet names: `ifso` for `&` and `ifnot` for `|`. (Since there was little collision with Forth words, I opted for the single-character versions.) You can use whatever emotional crutch feels the most appropriate to you, or none at all.

It was apparent that just falling out of the line (or colon definition) was inadequate for my needs. Most people aren't willing to factor their definitions to that level, nor should they have to. There was also the question of looping, restarting the execution at some previous point in the code.

Those operators `&` and `|` marked a point of departure, but there needed to be something (other than `:` and `;`) to mark the points of return. In other words, if `&` and `|` caused you to break out of the execution of the code, based on some condition, where do you land? Other languages have their endifs, thens, line terminators, etc. After serious deliberation, I decided to confiscate C's block delimiters `{` and `}`.

I know, I hear you saying, "Here is just another character trying to make Forth more C-like," but trust me, the icons are where the similarity ends. Years ago, someone suggested they look like profiles of faces, and I took it one step further by saying that, together, they resembled the faces, comedy and tragedy, of live theater.

With this worked out, conditionals `|` and `&` from which to branch, and layers of balanced control block delimiters, I had discovered how to lay down the `0BRANCH` control ops. The only thing left to determine was how to lay down the `BRANCH` ops responsible for looping and skipping the `ELSE` part of a conditional construct.

My solution was to use combinations of one or more carets (^) and lower-case v's (you can use upper case, if your system converts case automatically) to specify that execution was to float or sink that many control block levels. (Actually, "levels" is probably the wrong word here. You are actually "de-nesting" if you are thinking in terms of C, but what matters is whether you de-nest toward the top or the bottom of the program.)

Let me see if I can bring this all together for you in Figure One. You will notice later that sink 1 (v) is never needed: the default behavior is to fall out of the current block at the end. `&` and `|` serve the purpose of falling out of the current block before the end, based on a condition.

Some of you may find that a caret (^) is difficult to enter with your editor, especially if yours is patterned after the Forth, Inc. line editor, which uses it as a delimiter. Two alternatives exist. Presented in the order of my personal preference:

- Change your editor's delimiter to something else; my suggestion would be a line feed; this character shouldn't have too adverse an affect on any video updating and, since it's non-printable, you would be highly unlikely to want to enter it into your source code.

Or:

- Change the icon used for the float (^). You could use the tilde (~), which on most CRT's looks like a bubble, and that could be your memory aid: The more bubbles, the higher it will float.

Another thing you will

Figure One. UCS primitives.

Icon	Say	Description
`{`	comedy	Begins a new control block.
`}`	tragedy	Ends the most recently open block.
		(Note: these can nest to any depth)
`&`	ifso	Falls or "sinks" out of the current control block if the top of stack is false.
`|`	ifnot	Same as above if TOS is true.
`^`	float 1	Rises back to the top of the current control block.
`^^`	float 2	Rises back to the beginning of the control block before this one.
`^^^ ^^^^` ...		Logical extension of the above.
`vv`	sink 2	Falls through or "sinks" to the end of the second subsequent control block.
`vvv vvvv` ...		Logical extension of the above.

Kevin Haddock's first Forth—a FIG model adapted from CP/M—came from an OASIS users group. After switching himself and his customers over to MS-DOS and Unix, he worked in a network access control system done in native polyFORTH, a spinoff of the King Kahlid Airport in Saudi Arabia. On the side, he cannibalized some Commodore 64s and built a native fig-FORTH system on the C-64's motherboard, using a small daughterboard; this provided SCSI, RS-232, and Centronix capabilities almost entirely in software. Lately he has been finishing up a Forth written in C, not feeling that previous efforts to do this really captured the essence of Forth. He says, "Usually the inner interpreters were unconventional (e.g., case statements, jump tables, etc.) and provided little portability or comfort to Forthers seeking familiarity in an alien (e.g., Unix) environment."

UCS — F83 version.

```
\ UCS -- interpretative scanning
: @CHAR (S - c)   >IN @  BLK @ ?DUP IF  BLOCK + C@  ELSE
   TIB + C@  THEN ;

: ACCOUNT (S cnt chr - cnt')   DUP ASCII { = IF  DROP 1+
   ELSE  ASCII } = IF  1-  THEN  THEN ;

: >NEXT (S n)   0< IF  1  ELSE  -1  THEN  >IN +! ;

: I~ (S n)   -1 >IN +!   DUP 0< IF  1  ELSE  -1  THEN
   BEGIN  @CHAR  ACCOUNT  2DUP = 0= WHILE  OVER >NEXT
   REPEAT   2DROP ;

: I&|   BEGIN  @CHAR ACCOUNT ?DUP WHILE  -1 >NEXT  REPEAT
   1 >IN +! ;

\ Universal Control Structures
: { (S - a 0)   STATE @ IF  HERE 0  THEN ;  IMMEDIATE

: & (S a - a)  STATE @ IF   COMPILE ?BRANCH  HERE  SWAP ,  ELSE
   0= IF  1 I&|  THEN  THEN ;  IMMEDIATE

: | (S a - a)   STATE @ IF   COMPILE 0=  [COMPILE] &   ELSE
    IF  1 I&|  THEN  THEN ;  IMMEDIATE

: } (S a a)   STATE @ IF   BEGIN  ?DUP WHILE  DUP @  SWAP
   HERE SWAP !   REPEAT  DROP  THEN ;  IMMEDIATE

\ Universal Control Structures
: ` (S - a _)   BL WORD  NUMBER  DROP ;   IMMEDIATE

: ~ (S a1 a2 a1 a2 ... n)   STATE @ IF  COMPILE BRANCH  >R  SP@
   R@ ABS 2* 2*  +  R> 0< IF  HERE OVER @ ,   SWAP !
   ELSE  2+   @   ,  THEN  ELSE  I~  THEN ;  IMMEDIATE

: ~: (S n _)   CREATE  ,  DOES>  @ [COMPILE] ~ ;

: ~S: (S e s _ _ _ ...)   DO  I ~:  IMMEDIATE  LOOP ;

5 -4 ~S:  vvvvv vvvv vvv vv  ^ ^^ ^^^ ^^^^ ^^^^^

\ DUMP
16 CONSTANT WIDE
: .CHAR (S c)   { { DUP 32 <  |  DUP 126 >  | VV }
   DROP  ASCII .  }  EMIT ;
: .ASC (S a c)   {  OVER C@ .CHAR  1-  SWAP 1+  SWAP ?DUP & ^ }
   DROP ;
: .HEX (S a c)   {  OVER C@  0 <# # # #> TYPE  SPACE  1-
   SWAP 1+  SWAP ?DUP &  ^ }  DROP ;
: .ADR (S a)   0  <# ASCII : HOLD  # # # # #>  TYPE ;
: .DUMP (S a c)   { { KEY? | CR  BASE @ >R  OVER .ADR
   2DUP .HEX  SPACE  .ASC  R> BASE ! VV } ABORT } ;
: DUMP (S a c)   >R  { R@ & { DUP WIDE MOD ?DUP | WIDE }
   R@ MIN  R>  OVER -  >R  2DUP  .DUMP  +  ^ }  R> 2DROP ;

@CHAR returns the next char in the input stream
ACCOUNT -- given the current level count and char returns the
  adjusted count
```

(Continues.)

notice is that the branch operators (^ and v) and their derivatives (^^, ^^^, vv, vvv, etc.) will always appear just before tragedy (}). There may be some elegant way of combining the two, but I haven't found any. Also, since comedy and tragedy don't really take up any compiled memory, there is no real loss.

Some may argue that the universal control structures (UCS) are nothing more than an elaborate GOTO scheme. Albeit that some may use this tool to take Forth further down the path of obfuscation by creating even more poorly factored and deeply nested eyesores, I must defend the utility by saying that with the requirement to keep the control delimiters balanced, you would have a very difficult time branching wildly into the center of some block of code without clearly indicating your intentions.

I hope this utility will be a win-win situation for all users. The vertical-obfuscated-code people will be able to nest deeper than ever before (if that can be considered a win); the horizontal factorers can construct the structure to exactly what they want; and the poly-, cm-, and Pygmy-Forth types can get even wilder than before (this utility is great for passing control out of one word and into the middle of another, for example).

As you will see, there are actually two versions of each word, one for executing and another for interpreting. These are combined into one state-smart word where applicable; this may be one of the few places where a state-smart word may be excusable, since there is only a slight compile-time penalty and no run-time penalty.

A note of caution is in

order here: the interpreted version just does a string search for the balancing control delimiter, so remember: if you use braces—er, faces—in your words or in comments, always use them in balance. For instance, you could name a word `{FRED}` but try to avoid `{FRED` unless it is to be followed almost immediately with something like `FRED}`. This is the only minor un-Forth-like limitation on syntax, but I feel it is a small price to pay for the benefit received.

Possible uses for the interpreted version could be compilation scripts, non-time-critical routines, memory-critical programs, debugging with zero compile time, and my favorite: a late-binding, object-oriented system where methods could exist in different vocabularies that know how to deal with their own data types. You select the context by determining what kind of data object you are dealing with, then process the actual, reusable algorithm interpretatively from disk. With the old FIG-style hierarchical vocabularies, you could essentially build superclasses.

And now for the moment you have all been waiting for: on to the code!

Due to the drastically different behavior of the compiled and interpreted operators, I will describe them separately, even though some versions of Forth will bundle them into the same state-smart word.

It goes without saying that all control structures are compiler directives and, therefore, have precedence in whatever mechanism is appropriate for your Forth.

While compiling, UCS holds a pair of addresses on the stack for each open con-

```
>NEXT -- adjusts the input stream pointer based on the
  sign of the given number

I~ -- adjusts the interpretative pointer to the given control
   block.  Negative control block offsets move the adjust the
   pointer forward.   Positive or zero backward.

I&| -- adjust the interpretative pointer to the end of the
   current control block.

{ -- denotes the start of a control block.

& -- falls out of the current control block if given value is
   false

| -- falls out of the current control block if given value is
   true

} -- marks the end of a control block.  When compiling, this
   word resolves all the forward references for this block.
```

UCS — Pygmy version.

```
( Universal Control Structure   901127 KAH)   COMPILER
: {   ( - a 0)     \ BEGIN  0 ;
: &   ( a - a)    COMPILE 0branch  HERE dA @ -  SWAP , ;
: |   ( a - a)    COMPILE NOT   \ & ;
: }   ( a a)  BEGIN   ?DUP WHILE  DUP @  SWAP HERE dA @ -  SWAP !
              REPEAT  DROP ;

FORTH

comment:
UCS -- PYGMY VERSION
{ opens up a conditional structure.  When in doubt use a lot
   of these.  Make sure to close them.  Check your stack after
   compiling to make sure you got them all.
& takes an argument off the stack and if false branches to the
   end of the current conditional structure.  Does a boolean
   and but can be thought of as (and pronounced) 'if so'.
| does the same as above except branching when true.  Does a
   boolean OR.  Can be thought of as (and pronounced) 'if not'.
} resolves all the exit conditional branches for this structure
   and marks the exit location.  These must balance the open
   braces above. Check your stack after compiling when not sure

comment;

  ( Universal control structure    901127 KAH )    COMPILER
: `  ( _ - a)    32 WORD NUMBER ;

: ~ ( a1 a2 a1 a2 ... n _)    COMPILE branch
    DUP ABS   BEGIN  ?DUP WHILE  2SWAP  PUSH PUSH  1- REPEAT
    DUP PUSH  0< IF  HERE dA @ - SWAP ,  ELSE  OVER ,  THEN
    POP ABS  BEGIN  ?DUP WHILE  POP POP  ROT   1- REPEAT ;
```

```
: ^    0 \ ~ ;    : ^^    1 \ ~ ;    : ^^^    2 \ ~ ;
: ^^^^    3 \ ~ ;    : ^^^^^    4 \ ~ ;

: vv   -1 \ ~ ;    : vvv    -2 \ ~ ;
: vvvv    -3 \ ~ ;    : vvvvv    -4 \ ~ ;

FORTH

comment:
UCS -- PYGMY VERSION
` allows you to specify the branch direction and number inline
   while compiling, ie ` -5 ~  branches down 5 levels (not
   including the one you are in)

~ lays down an inline branch backward (for positive numbers)
  or forward (for negative numbers) the given number of levels.

^'s and v's are shorthand for forward and backward branches
comment;

( Interpretive universal control structures)    FORTH
: { ;    : } ;

: @CHAR ( - c)    { {   >IN @   BLK @ ?DUP &   BLOCK + C@   vv }
    TIB @ + C@ } ;

: ACCOUNT ( cnt chr - cnt')    { {   DUP   '{ = &   DROP 1+   vv }
    '} = &   1- } ;

: >NEXT ( n)    { { 0< &    1   vv }   -1 }   >IN +! ;

: \~ ( n)    -1 >IN +!    { { DUP 0< &   1   vv }   -1 }
     { @CHAR ACCOUNT   { 2DUP = | OVER >NEXT    ^^ } }   2DROP ;

: ~    \~ ;

comment:
UCS -- PYGMY VERSION

{ and } are just markers in interpretative mode

@CHAR returns the next char from the appropriate input stream

ACCOUNT returns a possibly adjusted brace level counter for
  the given char and counter

>NEXT adjusts >IN in the proper direction for the given level

\~ scans for the given number of uneven braces, backward for
   positive numbers and forward for negative
\~ is the compiliable version, ~ is the user interface

comment;

( Interpretive Universal Control Structures)
: & ( n)   { |   1   { @CHAR ACCOUNT ?DUP &   -1 >NEXT ^ }
     1 >IN +! } ;
```

(Continues.)

trol block. The top one points to a null-terminated thread of forward branch offset cells (the ones compiled right after the BRANCH and 0BRANCH ops) for the current control block. The second one holds the address of the beginning of the control block, to resolve the backward branches.

Comedy ({) just leaves the current dictionary address and the initial 0 on the stack.

`Ifso` (&) compiles the conditional branch op 0BRANCH, pushes the address of the offset cell following it (HERE), then compiles (comma's) the previous (possibly null) forward reference link there. This adds the next link in the forward reference chain.

`Ifnot` (|) is self-explanatory, just compiling an op to invert the test for `ifso`.

All the floaters and sinkers are just user interfaces to a word that digs down in the compile-time stack, either to link into the appropriate forward thread or to resolve a backward branch.

Tragedy (}) just tosses the control block start address left by comedy, then loops through the thread, resolving the forward branches until terminated by the null link.

The interpreted versions of comedy and tragedy do nothing except act as targets for character scanning.

In an actual installation, the character-scanning primitives (consisting of the words @CHAR, ACCOUNT, >NEXT, and \~, and even the loops in & and |) should be in machine code. I will leave it as an exercise to you, the reader, to work this out. Suffice it to say the code just scans the input stream pointer across the text until it finds the appropriate level of imbalanced open or closed braces. Also, you will note in the Pygmy version, this gives us our first live example of compiled UCS in action. I also leave it as an exercise to re-code the compilable part of UCS for metacompiling (and even your Forth nucleus, if you get so inspired). I'm not saying a Forth should not have the traditional control structures, just that it might be a more elegant solution defining them and Forth in terms of UCS, rather than the other way around.

Included in the listings are some quick examples: a case statement in the form of what could be a key/command loop, both in interpreted and compiled form; a memory dump utility; and a simple utility to create memory headers that will go out and interpretively load blocks, saving memory and compilation time (also with possible object-oriented implications as mentioned earlier).

As a final note, remember to keep an eye on your stack depth changes before and after compiling. The UCS have, and need, no compiler security. Too many comedies will grow your stack and too many tragedies will shrink it. Minimalists should already be familiar with this technique.

I hope you will have as much fun playing with, and improving upon, the UCS as I did discovering them. Enjoy!

```
: | ( n)  { &  1  { @CHAR ACCOUNT ?DUP &  -1 >NEXT ^ }
     1 >IN +! } ;

: ^   0 \~ ;   : ^^   1 \~ ;   : ^^^   2 \~ ;
: ^^^^   3 \~ ;   : ^^^^^   4 \~ ;

: vv   -1 \~ ;   : vvv   -2 \~ ;
: vvvv   -3 \~ ;   : vvvvv   -4 \~ ;

comment:
UCS -- PYGMY VERSION
These have basically the same meaning and usage as the
  compileable ones
comment;

( \ \REM \COMP ED: and SCAN)

DEFER \

: \COMP  CREATE  BLK @ ,  POP DROP  DOES> @ LOAD ;

: \REM   >IN @  64 / 1+  64 * >IN ! ;

' \REM IS \

: ED: ( _)   ' 3 + @ EDIT ;

: SCAN  ['] \COMP  [ ' \ 1+ ] LITERAL !
    THRU  ['] \REM  [ ' \ 1+ ] LITERAL ! ;

comment:
UCS -- PYGMY VERSION
\ during loading acts as a interpretative comment marker and
    during scanning compiles the following word which loads
    the affected screen
\COMP is the version that compiles the loading word
\REM is the comment (REMark) marker

Initially set the vector to remark

ED: <name> edits the screen for the loading word <name>
SCAN acts like thru except scanning and creating loading words

comment;

( Compiled sample key input/case loop )
: TEST
  { {
    CR ." ENTER A LETTER:"  KEY

    { DUP 27 = & ." BYE "  vvv }

    { DUP 'A = & ." ALPHA "  vv }
    { DUP 'E = & ." EDWARD "  vv }
    { DUP 'I = & ." IDA "  vv }
    { DUP 'O = & ." OCEAN "  vv }
    { DUP 'U = & ." UNION "  vv }
```

```
    CR  BEEP DUP EMIT ."  IS NOT A VOWEL! "
    }  DROP  ^
  }   DROP ;

comment:
UCS EXAMPLE -- PYGMY VERSION
An example of a compiled case structure
hit escape to quit

comment;

( DUMP )   16 CONSTANT WIDE
: .CHAR ( c)    { { DUP 32 <  |  DUP 126 >  |  vv }
    DROP   '.  }  EMIT ;

: .ASC ( a c)    { SWAP C@+  .CHAR  SWAP 1-  ?DUP &  ^ }
    DROP ;
: .HEX ( a c)    { SWAP C@+  0  <# # # #>  SPACE
     SWAP 1- ?DUP &  ^ }  DROP ;

: .ADR ( a) 0  <# ': HOLD # # # # #> ;

: .DUMP ( a c)   ?SCROLL CR  BASE @ HEX PUSH  OVER .ADR
    2DUP .HEX   SPACE  .ASC   POP BASE ! ;

: DUMP ( a c)   PUSH  { I & { DUP WIDE UMOD ?DUP |  WIDE }
    I MIN  POP OVER -  PUSH  2DUP  .DUMP +  ^ }   POP  2DROP ;

comment:
UCS EXAMPLE -- PYGMY VERSION
WIDE returns how many bytes to dump per line
.CHAR prints the given char if printable else prints an elipse.
.ASC prints the given count number of chars at the given addr
.HEX prints c hex bytes at the given addr
.ADR prints the given address
.DUMP prints one line of the dump for the given addr and count

DUMP dumps c bytes of memory at the given addr.

comment;

\ TEST  interpreted sample key input/case loop
{ {
  CR ." ENTER A LETTER:"  KEY

  { DUP 27 = & ." BYE "  vvv }

  { DUP 'A = & ." ALPHA "  vv }
  { DUP 'E = & ." EDWARD "  vv }
  { DUP 'I = & ." IDA "  vv }
  { DUP 'O = & ." OCEAN "  vv }
  { DUP 'U = & ." UNION "  vv }

  CR  BEEP DUP EMIT ."  IS NOT A VOWEL! "
  }  DROP  ^
}   DROP
```

(Letters, from page 7.)

ded systems domain. Only time will tell if newcomers to Forth prefer Frank's meta-compiled Pygmy Forth, or Dr. Ting's assembly-language eForth.

Brad Rodriguez
B.Rodriguez2 on GEnie

Forth on the Macintosh

Laughing Water of Helena, Montana, told us about his becoming a Forth defector on the Macintosh in the May/June issue of *Forth Dimensions.* It is, in many ways, his article that incites the writing of this one.

I have used Forth on the Macintosh since the early days of the 128K Mac. I have had a running application going and in daily use—developed in Mac4Th at first and later finalized in MACH2 Forth—for about five years now. It controls a studio 1/4" tape recorder, greatly facilitating the logging and locating of sound effects, atmospheres and daily dialogue takes in my work as film sound recorder and dubbing editor. But I, too, have during the last year or so become somewhat of a Forth defector, using more and more time with other Macintosh developing environments, MPW to be more precise. Note I use the word environment and not language.

A quote from Laughing Water: "Besides, the Mac listens better when I talk Pascal." I read: "The Pascal environment he uses is better than the Forth one, or at least better understood."

With today's Mac programming choices—MPW

(Pascal, C, C++, MacApp, Assembler), MDS Assembler, Language Systems Fortran, MacFortran, Prograph, Smalltalk, Think C, TLM Pascal, True Basic, Lisp, Modula2, Simula, MacForth (still on sale), Mach2 Forth (no more advertised), Neon (become public domain), MasterForth (not supported for a long time), Pocket Forth, Hypercard, and... as well as the many dBase-like languages like Omnis, 4th Dimension, etc.—sure the Mac "speaks many languages," but the environments are "oh, so different."

Macintosh programmers are busy generating standalone applications and adhering rigidly to Apple's programming guidelines. Forth is not concerned with that. (How could it be?)

The majority of Forth code exchanged within the Forth community in journals like *Forth Dimensions* and the *Journal of Forth Application and Research* doesn't deal with standalone applications, and only a very few times do they approach the subject of a Mac-like graphical user interface. Never, I believe, have they dealt with Macintosh issues in particular.

The Macintosh community is, naturally, debating Macintosh issues with all it's virtues and quirks. When *Mactutor*'s Forth column was stopped in favor of C++ and MacApp, the Macintosh's Forth community lost its last meeting platform. Whatever happened to the MacForth user group? The last newsletter I got was sometime in late 1986!

The Forth community is, naturally, busy debating Forth. And it is mostly machine independent.

The Forth compiler and interpreter is a powerful tool, in itself, that is always present. This is perhaps the major and most important difference between Forth and any other language, and is immensely hard to explain to non-Forthers. A lot of Forth's power lies precisely in its accessibility: the ability to extend the compiler and interpreter, to add to it, to use or abuse it (refer to Defector's reference to anarchy) within new definitions. The fact is that the processor is there for you to use, directly from an interactive console or via some prior compilation.

On the Mac, we are expected to program so that, ultimately, the Forth layer will not be present. No more definable words, no more vocabularies, no more Forth interpreter! Some of Apple's programming guidelines also affect the available Forths directly. For example: not to use any of the 68000 trap mechanisms, and not to use in-line variables in code segments (due to future instruction caching). Words like

There is no reason why the vast programming utilities of MPW could not become available to the Forth environment.

(Continued on page 30.)

Neural Network Words

Tim Hendtlass
Hawthorn, Victoria, Australia

THE LAST OF TWO PARTS

In part one of "Neural Network Words," a number of words were developed to build and train neural networks. Words are now presented to save a network layer to disk. Most networks will learn from real life data rather than from an artificial example, such as the XOR relationship used in part one. The advantages of scaling the real life input and output values to suitable internal values for the neurones used was also demonstrated in part one.

Working out scaling factors is tedious, at best, and words are now presented to read a data file, calculate the required scaling factors, and save these to disk. Finally, a general-purpose word to train a network is presented. It is customised to a particular application by assignments made to four deferred words. An example that uses these words is given.

The more complicated the relationship to be learned, the more neurones will be needed.

ERRATA

In the first part of this article (FD XIII/2), we erroneously printed an incorrect figure. In that issue, the Forth file printed as Figure Eight (beginning on page 20) actually contains words related to disk I/O, which is discussed in this *installment. Please refer to it for that code. In turn, the file NNDEMO.SEQ, which should have been printed last time as Figure Eight, appears herewith. We apologize to the author and to any readers who were inconvenienced.*

The author also wishes to note that in DISKNN.SEQ (the file which appeared last time as Figure Eight), he used a couple of outdated word names in the opening comment: `NL->DISK` *instead of* `SAVE-LAYER`, *and* `DISK->NL` *instead of* `LOAD-LAYER`. *—Ed.*

Words to copy a network layer to and from disk

`SAVE-LAYER`
This word saves a neural layer to disk. The address of the layer to save must be on the stack on entry, and the name of the file to create must immediately follow `SAVE-LAYER` on the current input line. If a file by this name already exists, the user is asked for permission to overwrite it.

`READ-LAYER`
This word restores a neural layer from disk. The layer must have been created before the word is called. The address of the layer to copy to must be on the stack on entry, and the name of the file to read must immediately follow `READ-LAYER` on the current input line. No check is made to see that the layer being read to has the same dimensions as the layer on disk.

Words to read a data file

Data files used by this software are simple ASCII files, with one set of input values and one set of output values per line. No explicit provision is made in this version for comments, although anything on a line after the specified number of inputs and outputs will be skipped over. The variable `N-IN` specifies the number of inputs to the whole network, the variable `N-OUT` the number of outputs from the whole network. `READ#` reads one number from the current input and scales it into internal form.

The convenient word `PROCESS-FILE` saves information about what we were doing and moves up one handle on the handle stack. It then opens a new file, processes it, closes it, and returns to what we were doing when the `PROCESS-FILE` word was encountered. `PROCESS-FILE` must be followed directly by the name of the file to use, and processing continues immediately after the filename. The actual processing done is controlled by the deferred word `(PROCESS-FILE)` which can be set to anything at all. `SET-SCALES` and `TEACH-NETWORK` (see below) differ mainly in what `(PROCESS-FILE)` is equated to.

Scaling words

`SET-SCALES` reads the file specified immediately following this word. It finds the maximum and minimum input and output numbers, and works out the scaling factor to scale the real life input and output values to the range from -0.5 to +0.5, which suit the non-linear transform in use. After the scale factors have been set, the words `SCALE-IN#` and `SCALE-OUT#` scale an input or output number. The word `UNSCALE-OUT#` returns an output from internal representation to real life units. The scaling factors can be saved to disk and restored from disk with `SAVE-SCALES` and `READ-SCALES`, respectively. Each of these words must be immediately followed by the name of the file to use.

The generic training word

`TEACH-NETWORK` is the top generic training word. It expects the number of training passes to be on the stack on entry, but if the stack is empty it defaults to 1000 cases. It must be followed by the name of the data file to use. It does not establish the scale factors—these must have been just calculated, or a previously generated set must have been read from disk.

`TEACH-NETWORK` contains four deferred words, and these must be pointed to the routines for the specific task in hand. The deferred words, and their functions, are:

`DO-INIT`
This may do any initialization of disk, display, or anything else required.

`FORWARD-WORD`
This performs one evaluation pass of the network, updating the internal activation of each output. It expects the inputs, already scaled, on the stack on entry. Once a network is trained, this is the basic word for using it in recall mode.

`TRAIN-WORD`
Given a set of scaled outputs on the stack, this word must calculate the errors in the outputs and back-propagate these to the network inputs, updating the weights as it goes. It leaves nothing on the stack.

`DISPLAY-INFO`
This is a word to display any information about the state of the network that may be desired. It is called after each training example.

With these words, `TEACH-NETWORK` will train for the specified number of cases or until a key is pressed. Figure One shows the code for the words described above. *[See Errata on page 17; the Figure One referred to here was printed as Figure Eight in the last issue.—Ed.]*

Example

Figure Two shows an example that uses `TEACH-NETWORK`, and Figure Three provides a suitable data file. Once again, the exclusive-or relationship is used so that the file is short, but any other could be substituted. The more complicated the relationship to be learned, the more neurones will be needed. If too few are used, the network either will not stabilise or will learn only the general features of the relationship and fail to learn the detail. Of course, this ability to generalize may be of considerable interest in some situations. The error display is very simple, just printing the error for the four examples per line; since there are four possible examples, only this provides a simple way to see how the network is behaving. Alternatively, a simple four-graph plotting routine would make trends easier to observe.

Figure Three. A small sample data file (XOR.DAT).

```
0 0 0      \ input input output
1 0 1      \ input input output
0 1 1      \ input input output
1 1 0      \ input input output
```

Conclusion

The words presented here extend those in part one. Together, they provide a toolkit for constructing non-linear feed-forward networks and for training them by back propagation. They demonstrate again the power of Forth to produce a special language for a task. Following the approach used here, words could simply be produced to implement other types of neural layers, such as self-organising Kohonen layers. By standardising on passing data between layers on the stack, networks of arbitrary complexity can be readily constructed, trained, and used.

Tim Hendtlass has used Forth for years, and teaches it to about 100 students per year at the Swinburne Institute of Technology. He relates, "Roughly half the class are taking Computer Science as their co-major and, by the time they get to me, have one year of programming and several (nearly) working programs under their belts; they are sure they know it all. The other half are taking Medical Biophysics and generally are not sure what a computer is, are sure they don't like it, and have a fear that touching the keyboard will result in the machine taking over their minds.

"Despite this difference in background, within about one semester they are all controlling instruments in the laboratory using multi-tasking and hardware interrupts, all written in Forth. They like it, to their collective surprise. I would be delighted to hear from anyone else teaching Forth as a primary language of choice and as a vehicle to let students achieve some non-computing-related objective (scientific instrumentation systems, in my case).

"I have developed a series of experiments that I would be glad to share, and I would welcome hearing about others' experiences. Write to me in care of the Swinburne Institute of Technology, P.O. Box 218, Hawthorn 3122, Australia; or fax to 819 5454."

Figure Two. An example using TEACH-NETWORK (NNDEMO2.SEQ).

```
  2 n-in ! 1 n-out !              \ define number of network inputs and outputs
  1 3 layer TOP
  top initialize                  \ top (output) layer
  3 2 layer BTM
  btm initialize                  \ bottom (input) layer
  clipping off

: -->    ( -- )
  btm compute                     \ use stack data and evaluate the bottom layer
  btm get-outputs                 \ load bottom's output to stack
  top compute                     \ use bottom layer outputs, evaluate top layer
;

' --> IS FORWARD-WORD

: <--    ( On On-1 .... O1-- )
  top calc-load-errors            \ calculate and load the errors
  s0.3 top train                  \ train top layer, learning rate of 0.3
  btm load-errors                 \ load back propagated errors
  s0.8 btm train                  \ train bottom layer, learning rate of 0.8
  n-in @ 0 do 2drop loop          \ lose back-propagated errors @ network inputs
  1 icount +! ;                   \ increment itteration count

' <-- IS TRAIN-WORD

: GET-ERROR    ( n  -- S# )       \ get nth output UNSCALED error
  dup get-oeadr 2@                \ get the internal error
  rot get-intact 2@ s'fn(x) s/    \ convert to external form
  o-s 2@ s/                       \ and unscale it
;

: .ERROR                          \ show errors each set of inputs
  icount @ 1- 4 /mod swap 0= if   \ time for an new error line?
    crlf ." Set # " 1+ 4 .r ." - errors "
  else drop then 1 get-error s.   \ print abs error
;

' .ERROR IS DISPLAY-INFO

: MISC-INIT 0 icount ! ;

' MISC-INIT IS DO-INIT

\ EXAMPLE
  set-scales xor.dat              crlf .(  Conversion scales computed ) crlf
  save-scales xor.sca             .(  Scales saved) crlf
  500 teach-network xor.dat       crlf .(  Network teaching terminated)
  top save-layer xortop.lyr       .(  Top layer saved) crlf
  btm save-layer xorbtm.lyr       .(  Bottom layer saved) crlf
```

NNDEMO.SEQ code pertaining to Part One.

```
\ Four examples of simple artificial neural networks.   Tim January 1990
\ ************************************************************************
comment:
These examples all use simple non-linear feedforward networks and are trained
by back propagation.  For simplicity, they use very simple inputs.  Examples
2 to 4 all learn to impliment the exclusive or relationship, which practically
would never be generated this way!  Needs BASICNN to have been loaded, the
line below will check and load this file if it is not already loaded.
comment;
\ ************************************************************************
                ( ===> ) NEEDS BASICNN.SEQ ( <=== )
\ ************************************************************************
anew program

\ Example One, single layer learning two relationships.
  1 2 layer TEST             \ build network with 2 inputs and 1 output

: EX1 ( -- )                 \ the word that puts example 1 together
  test initialize            \ initialize layer
  1 icount !                 \ initialize itteration counter
  clipping off               \ clipping not required
  begin
  crlf ." Pass " icount @ .
  S0.5 S0.5                  \ first use inputs of 0.5 and 0.5
  test compute               \ do one forward pass
  test get-outputs           \ output to stack
  2dup ." Output " s.        \ print result we got
  S-0.5 2swap d-             \ calculate error, result should be -0.5
  test load-errors           \ load the error
  S0.8 test train            \ update weights using a learning rate of 0.8
  2drop 2drop                \ only 1 layer,backpropagated errors not needed
  S-0.5 S1                   \ second use inputs of -0.5 and 1
  test compute               \ do one forward pass
  test get-outputs           \ output to stack
  2dup ." Output " s.        \ print result we got
  S0.5 2swap d-              \ calculate error, result should be 0.5
  test load-errors           \ load the error
  S0.8 test  train           \ update weights using a learning rate of 0.8
  2drop 2drop                \ don't need the backpropagated errors
  1 icount +!                \ bump the iteration counter
  key? if key drop exit then \ keep going until we get bored
  again
;

\ Example Two, two layers, XOR, standard input
1 3 layer test2                 \ 1 output, 3 inputs    The top layer
3 2 layer test1                 \ 3 outputs, 2 inputs   The bottom layer
defer docases                   \ the difference between examples 2 3 and 4
defer top-layer
defer bottom-layer
: One-pass ( correct-output input2 input1 -- error )
  test1 compute                 \ forward process the bottom layer
  test1 get-outputs             \ get its outputs ready for the next layer
  test2 compute                 \ forward process the top layer
  test2 get-outputs             \ get its outputs ready for the error calc
  2dup s. d-                    \ print actual output, compute error
```

```
   test2 load-errors            \ load into test2 ready for training
   s0.3 test2 train             \ update layer 2 weights, learning coeff 0.3
   test1 load-errors            \ load into test1 ready for training
   s0.8 test1 train             \ and in layer 1, learning coeff 0.8
   2drop 2drop                  \ lose errors backpropagated to inputs
 ;
 : Ex2-docases
     S0 S0 S0 one-pass          \ do 0 0 case
     S1 S0 S1 one-pass          \ do 0 1 case
     S0 S1 S1 one-pass          \ do 1 1 case
     S1 S1 S0 one-pass          \ do 1 0 case
 ;
 : (Ex2-4)
   1 icount !                   \ initialize itteration counter
   clipping off                 \ clipping not required
   begin
     crlf ." Outputs "
     docases
     ." after pass " icount @
     1 icount +!
     key?                       \ user want anything?
     if
       crlf top-layer .layer    \ if so show them the layers
       crlf bottom-layer .layer
       key upc ascii Q =        \ they want to quit?
       if exit                  \ if so get out of here
       else begin key?          \ if not wait..
            until key drop crlf \ ..until they tell us that we may go on
       then
     then
   again
 ;
 : Ex2
   test1 initialize             \ initialize our...
   test2 initialize             \ ...two layers
   ['] test1 is bottom-layer
   ['] test2 is top-layer
   ['] ex2-docases is docases
   (ex2-4)
 ;

 \ Example Three, two layers, XOR, scaled inputs and outputs
 : Ex3-docases
   S-0.5 S-1 S-1 one-pass       \ do scaled 0 0 case
   S0.5  S-1 S1  one-pass       \ do scaled 0 1 case
   S-0.5 S1  S1  one-pass       \ do scaled 1 1 case
   S0.5  S1  S-1 one-pass       \ do scaled 1 0 case
 ;
 : Ex3
   test1 initialize             \ initialize our...
   test2 initialize             \ ...two layers
   ['] test1 is bottom-layer
   ['] test2 is top-layer
   ['] ex3-docases is docases
   (ex2-4)
 ;
```

(Continued on next page.)

```
\ Example Four,two layers, XOR, special geometry,scaled inputs and outputs
1 3 layer test4               \ 1 output, 3 inputs    The top layer
1 2 layer test3               \ 1 outputs, 2 inputs   The bottom layer
: MOD-ONE-PASS ( correct-output input1 input2 -- error )
  4dup test3 compute          \ copy the inputs, use one copy on test3
  test3 get-outputs           \ get the output from the bottom layer
  test4 compute               \ other copy + test3 output = input to test4
  test4 get-outputs           \ get the actual output from the top layer
  2dup s. d-                  \ print actual output, compute error
  test4 load-errors           \ load the error into the top layer
  s0.3 test4 train            \ update weights in top layer
  >r >r 2drop 2drop r> r>     \ lose errors propagated back to the inputs
  test3 load-errors           \ load the  error at output of test3
  s0.8 test3 train            \ now update bottom layer
  2drop 2drop                 \ lose errors backpropagated to inputs
;
: Ex4-docases
  S-0.5 S-1 S-1 mod-one-pass   \ do scaled 0 0 case
  S0.5  S-1 S1  mod-one-pass   \ do scaled 0 1 case
  S-0.5 S1  S1  mod-one-pass   \ do scaled 1 1 case
  S0.5  S1  S-1 mod-one-pass   \ do scaled 1 0 case
;
: Ex4
  test3 initialize             \ initialize our...
  test4 initialize             \ ...two layers
  ['] test3 is bottom-layer
  ['] test4 is top-layer
  ['] ex4-docases is docases
  (ex2-4)
;
```


Advertisers Index

A message from the President of the Forth Interest Group

President's Letter

Organization, Issues, Actions, and Explanations.

Organization

Business Group

Summary of recent business meeting minutes for July 21, 1991 (full minutes of the Business Group are available upon request):

Membership dues (correction): In the last *FD*, I stated that the dues for the new student membership were $24 per year. That was one of the figures suggested, however the correct figure that was approved by the business group was a student membership of $18 per year.

ADC Reports: There are now 1511 FIG members to date. This is an increase from last year at this time. This turn-around has been a long time in coming, and I expect the upward trend to continue.

Board of Directors

The nomination process for the Board of Directors has taken place. According to the FIG by-laws, a nomi-

(Continued on page 26.)

Who would complain if Forth were required in their project?

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

	4/30/90	*4/30/91*	*Change**
ASSETS:			
Current Members	2106	1831	-275
Current Assets:			
Foothill Bank, Money Market	15,925.38	24,865.91	8,940.53
Foothill Bank, Checking	636.43	700.14	63.71
Pending Foreign Clearing	-102.00	51.67	153.67
Returned Checks Pending	0.00	72.00	72.00
FORML, Money Market	15,894.11	16,916.46	1,022.35
FORML, Checking	1,926.01	1,236.64	-689.37
Total Current Assets:	34,279.93	43,842.82	9,562.89
Inventory:			
Inventory at cost	34,147.53	26,601.17	-7,546.36
Total Inventory:	34,147.53	26,601.17	-7,546.36
Other Assets:			
Deposit, United Parcel Service	200.00	200.00	0.00
Second Class Postal Account	156.31	192.41	36.10
Accounts Receivable	3,166.20	2,099.00	-1,067.20
Total Other Assets:	3,522.51	2,491.41	-1,031.10
TOTAL ASSETS:	71,949.97	72,935.40	985.43
LIABILITIES:			
Sales Tax due	33.89	35.58	1.69
FD Dues Alloc to future months *FD*	37,487.30	41,518.51	4,031.21
TOTAL LIABILITIES:	37,521.19	41,554.09	4,032.90
Financial Reserve:	34,428.78	31,381.31	-3,047.47

** Increase = +*
Decrease = -

Ada Multiprocessor Real-Time Kernel

Hoyt A. Stearns, Jr.
Phoenix, Arizona

A fully pre-emptive, priority driven, multiprocessing real-time kernel is required in certain applications (process control, in this case). After evaluating several schemes, I decided that the Ada rendezvous method is simple, straightforward, and elegant. Ada multitasking will work on any configuration of multiple processors that can communicate, or on a single processor. A single task can be localized to a particular processor, or can be distributed among many.

The listing is the kernel for a system with three processors. In the actual implementation, one of the three processors is of a completely different type, which is handled by having two code fields in each Forth word. All processors share the same memory.

Having a processor of a different type added some complications in the listing not normally necessary—such as the constant `WS` for word size—since the processors were of different addressability.

The kernel works fine on a single-processor system, in which case all references to semaphores may be removed.

Memory is allocated from the top down, starting at the contents of variable `RAMPTR`. First the system variables are allocated, then task control blocks (TCB's), as tasks are registered to the system with `REGIS`. Each TCB has three link fields, one for each processor, so a task may be registered to be scheduled among one to three processors.

Although this kernel implements the Ada rendezvous system, the syntax is, of course, different.

The Ada selective-wait structure is invoked in this system with the word `SELECT` (S bit_mask timeout_value – sender &msg entry_#), where each 1 bit in bit_mask represents an Ada entry that is open (`ADA ACCEPT`) and the timeout_value is how long in clock ticks to wait for a message on one of the open entries (`ADA DELAY`). The entry_# return parameter is which entry the message came in on, or the total number of entries on this task if it times out. The &msg is where to get data if this is an Ada :in message, or where to copy the data if it is an Ada :out message.

The sender parameter is the address of the caller's TCB, and is used as a parameter to the word `RELEASE`, which is equivalent to the Ada END_SELECT construct.

Ada entry calls are implemented here with the word `CALL` (S msg...timeout_value entry_# TCB_addr – msg... status) where timeout_value is how many clock ticks to wait for receipt of the message before returning with #call_expired status. The other parameters are which entry on which task to call. The message parameters must be explicitly dropped after the call, since `CALL` has no way of knowing how big the message is.

Other functions available are pure `DELAY` and `IXMIT`, which sends a message without stalling the caller (mainly used to send messages from interrupts). In Ada, interrupt messages are received by tasks just as if

Ada's multitasking works on multiple processors that can communicate, or on a single processor.

they were from another task.

Improper structuring of tasks in a system may easily result in deadlocks. The word DEDLOK may be invoked periodically to break any deadlocks by returning the parameter #BROKEN to one of the CALL's involved. This shouldn't be necessary in a properly programmed embedded system.

There should be a MASTER task—registered to only one of the processors in the system—which registers the rest of the tasks, then calls slave tasks on the other processors, which start up their own schedulers.

Interrupts were handled in the target system by revectoring NEXT in the variable NP (next pointer) on a particular processor to high-level code to post an IXMIT message, then revectoring NEXT back to normal. NEXT may also be vectored for tracing.

There is an ancillary task, SYSTAT, which takes a snapshot of the entire system and prints a report of task status, message queues, run-time statistics, etc.—but that is for another article.

Ada listing.

```
Scr # 1
\ load screen  real time kernel                        has063091
decimal
variable ramptr sp0 @ 2000 - ramptr ! \ top of tcb space
(S d addr--)
: d+!   dup >r 2@ d+ r> 2! ;
: umin   (S n1 n2--n3) 2dup u> if swap then drop ;
2 constant ws
4 constant dws \ word size--addr increments/word
vocabulary kernel kernel definitions
2 28 thru
 \ forth definitions

Scr # 2
\                                                       has063091
: access   dup create , does> @ + ;
: sys   dup create , does> @ ramptr @ + ;
\ receiver definitions
1 constant open   2 constant refus   128 64 + constant pused
64 constant used   32 constant rsemaphore
0 access rlist  ws + access rstate ws + constant rcvrsz
\ task definitions
1 constant selbt   2 constant endobt   4 constant delbt
128 constant stkbt  64 constant suspbt 32 constant tsemaphore
16 8 or constant proc_lock
0 constant #call_ok  2 constant #broken
1 constant #refused 3 constant #call_expired  0 constant id

Scr # 3
\                                                       has111690
0  sys event
dws -  sys clock  dws - sys start_time
ws 3 * - : up0   [ dup ] literal id + ramptr @ +  ;
ws - ramptr @ + constant first_tcb_top
0 up0 !
(S bits addr--)  : set tuck @ or swap ! ;
                 : reset tuck @ swap not and swap ! ;
(S word true false--flag)
: logic   over or -rot xor and 0= ;
: up@    up0 @ ;    : up!  up0 ! ;
```

(Code continues.)

(President, from page 23.)

nating committee consisting of two members of the Board of Directors (John Hall and Dennis Ruffer) was appointed. After a search for candidates, three were selected to fill the vacant positions. They are Mike Elola, current board member; Nick Solntseff, southern Ontario FIG Chapter; and Jack Woehr. Since there were insufficient write-in petitions (25) for any other candidate, an election by ballot by the membership is not required. A unanimous vote for the nominees will be cast by the Secretary at the annual meeting of the Board of Directors in November. These candidates will submit personal statements in the November-December issue of *Forth Dimensions.*

Issues, Actions and Explanations

Treasurer has prepared review of FIG financial situation. The treasurer has prepared, and the Board has approved, the financial statement presented in this *FD*. In explanation, April 30 is the end of FIG's fiscal year and the figures reflect those dates. The form is similar to a "Statement of Change in Financial Position" changed to reflect the year-ends for 1990 and 1991 with comparison. This form shows Assets and Liabilities, and the differences. Most categories are clear, with the exception of the "*FD* Dues Alloc. to future months *FD*" and "Financial Reserve." "*FD* Dues Alloc to future months *FD*" is money received by FIG, budgeted to complete a member's year of *Forth Dimensions* (this is the amount FIG is obligated to put aside to produce fu-

```
Scr # 4
\ low level                                            has062991
\ message definitions and call_frame definitions
0 access mlink  ws + access dest  access cdest
ws + access dest_rcvr access centry
ws + access sender  access cdelay  dup constant csize
ws + dup constant mdata  constant msize
0 : link   dup up0 <> if [ rot dup ]  literal
   + id + [ -rot ] then ;
ws 3 * + access nrcvr  ws + access priority ws + access tstate
  ws + access maxtim  ws + access actime dws + access urp
  ws + access usp ws +  access usp0 ws + access urp0
  ws + access uip ws + access unp  ws + access totime dws +
  access spmin ws +  access rpmin ws + access splim ws +
  access rplim ws +  access tname  2 dws * + access rcvrs
  ws + access _dp
  ws + constant tcbase_size


Scr # 5
\ user functions                                       has062991

(S  rcvr_addr--)  \ mlink must be 0 before free
: rsem_clr  rstate rsemaphore swap reset ;
(S rcvr_addr--) \ wait for semaphore, then set it
: ?rsem   rstate begin dup @ rsemaphore and 0= until
  rsemaphore swap set ; \ must be done under lock
: idlock ( semaphore to lock a processor's task list) ;
 : idunlock ;
: di   ( disable interrupts on this processor) ;
: ei   ( enable interrupts on this processor) ;

: ?tsem ( test and set a task linking semaphore) ;
: tsem_clr ( clear tsemaphore) ;
: sys_stks   ( set stack pointers to system area) ;
: 'next  ( address of "next" in target for NP, the next vector);


Scr # 6
\                                                      has062991
(S up -- f, true if activation time)
: ?actime   actime 2@ ( di)  clock 2@  ( ei) d- nip 0< ;

(S start_addr number item_size--item_size,+or- end_a start_a)
: <b   dup >r * dup >r over +  1 r> ?negate - r> -rot swap ;

(S up offset  -- size end start) \ setup loop over receivers
: rbnds   over rcvrs + swap nrcvr @ rcvrsz <b ;

(S up bit_mask--)
: sbclr swap 0 rstate
  rbnds do over i reset dup +loop 2drop ;
(S rcvr_addr--flag, open with msg)
: ?msg  dup dup >r ?rsem dup rstate @ open and 0<>
  swap rlist @ and r> rsem_clr ;
```

```
Scr # 7
\                                                  has062991
(S data up--)
: tpush   usp dup @ ws - dup rot ! ! ; \ push parm to up task
(S up rcvr_addr--rcvr_number)
: ra>r#   swap rcvrs - rcvrsz / ;
(S up rcvr_number--rcvr_addr)
: r#>ra   rcvrsz * swap rcvrs + ;
(S rcvr_addr--)
: rcv_msg   dup dup ?rsem rlist dup di @ tuck @ swap ! ei
  -1 swap ! ( rec'd) on rsem_clr ;
(S rcvr_addr--msg_addr)
: rcv>msg   rlist @ ;


Scr # 8
\                                                  has062991
(S rcvr_addr--)
: mark_used   rstate pused swap set ;
(S msg_addr--)
: clr_sender_delay    sender @ tstate delbt swap reset ;
(S &tstate--tstate flag)  \ True if not...
                          \ ...suspended, tsem=0, stk ok
: ?runnable  dup @ swap ?tsem over [ stkbt suspbt or ] literal
  and or 0= ;
(S --up')
: robin?   up@ dup link @ swap priority @ over priority @
  <> if drop up0 @ then ;
: setnxt (S up--) tsemaphore over tstate ! di up! ei
  clock 2@ start_time 2! ;


Scr # 9
\                                                  has112590
(S rsize end_addr rcvr_addr--rsize end_addr rcvr_addr'|false)
: find_msg   >r >r r@ swap false swap r> r> do i ?msg if
  nip i swap leave then  dup +loop -rot ;
(S up msg_addr--)
: msgexi   2dup sender @ swap tpush  2dup mdata + swap tpush
  dest_rcvr @ swap tpush ;
(S up rcvr_addr--)
: gotsy   over  open sbclr dup mark_used dup rcv>msg
  swap rcv_msg dup clr_sender_delay   msgexi ;
(S up--data)
: tpop   usp dup @ @ ws rot +! ;
```

ture issues of *FD*). "Financial Reserve" is the difference between Assets and Liabilities, and indicates the state of FIG if all Assets were liquidated to pay all Liabilities. It is an indication of the health of FIG.

Complete financial reports are available in the FIG offices and are open for inspection.

Public relations. For the last several months, the focus of the Business Group has been on publicity for Forth and FIG. There are several lines of attack that we are preparing to take. In the past, our approach has been to try to encourage individual programmers to look at Forth and, when they do, to use FIG as a resource. We are shifting the emphasis of exposure toward what I call, for lack of a better term, the "mid-level manager"—the people who control the direction of projects, who have some control of a budget, and who want to get the most product for the money they control. The emphasis will be on the innovation that is going on in Forth, the fact that there are people ready and available to help with their projects, and the wealth of information that is already available. These are the exact people about whom I have heard the complaint, "If only these people understood what could really be done with Forth, my life would be a lot easier." These are the people that I have heard complain, "I would like to use Forth, but where am I going to find enough of the right people to produce and maintain my project?" Two specific types in this group, to give you a better feel of what I mean by mid-level manager, are project leaders, at all levels of industry; and researchers in R&D organizations, whether universities or R&D divisions of larger companies.

How do we entice them?

The primary way will be with articles in all appropriate trade publications about the innovative ways that Forth is being used to solve current problems. Highlighted will be individuals or teams or applications or Forth vendors' products or standards or places where Forth already is used but the fact is generally unknown. The emphasis will be on innovative ideas and innovative people!

Second, by volume we let them know that they have always been surrounded by Forth and that the idea of using it is not novel—in fact, that it is required and that there have always been people near them who are willing to help.

Third, that FIG is one of their conduits for information about Forth, whether it is for literature, training, people, or direction toward vendor products. We will emphasize all the Forth resources available to make them and their projects successful.

How do we do all this? This is a multi-path attack, and one of the places it starts is with you!

1. If you will soon be writing a technical article about your project, you will see an article in the next *FD* about the style and approach that FIG would like you to add to your article. Horace Simmons has been compiling a list of the publications that will be the vehicles for this endeavor.

```
Scr # 10
\                                                     has111990

(S rcvr_addr--used_flag)
: ?used   rstate @ used and ;

(S rsize end_addr rcvr_addr--rsize end_addr rcvr_addr'|false)
: &unused   >r over r> + 2dup u> if  begin
  find_msg dup dup if ?used then
  0= until else drop false then ;

(S up--up rcvr_size rcvr_end_addr rcvr_addr|false)
: first_msg   dup 0 rlist rbnds find_msg ;

(S up rcvr_size rcvr_and_addr rcvr_addr--up rcvr_addr)
: acquire_used   dup >r &unused dup if r> drop nip nip
  over swap else 2drop drop dup used sbclr r> then ;


Scr # 11
\                                                     has062791
(S  &mlink &rlist--)
: msg_unlink  dup dup >r ?rsem rlist  2dup @ <> if @ 2dup mlink
   @ <> if mlink @ then then swap @ swap ! r> rsem_clr ;

(S &msg_frame--&rlist) \ precursor to find predecessor to msg
: msg>rlist   dup dest @ swap dest_rcvr @ r#>ra rlist ;

(S up--)
: cancel_msg   usp @ mdata - dup mlink
  swap msg>rlist msg_unlink  ;

(S up--up )
: caltxm   dup cancel_msg #call_expired over tpush  ;

: texi  (S --)   r> drop up@ uip @ >r ;


Scr # 12
\                                                     has112690
(S up--up)
: seltxm   dup open sbclr dup nrcvr @ over tpush ;

(S up state--up )
: time_activate   dup selbt and if drop seltxm else
  endobt and if caltxm  then  then  ;

(S up state--up true|false)
: timechk   over >r dup delbt and r> ?actime and
  if time_activate true else drop false then ;

(S up--up true | up false)
: inscan   first_msg dup if dup ?used if acquire_used else
  nip nip then over swap  gotsy true else nip nip then ;

```

```
Scr # 13
\                                                      has062991
(S up--up' true|false)
: look   dup tstate  ?runnable if dup
  if tstate up  dup selbt endobt logic
  if swap inscan if nip true else swap timechk then
  else timechk then 0= then else drop true then ;
(S up--)
: do_task   dup setnxt dup urp @ rp! dup urp0 @ rp0 !
  dup usp0 @ sp0 !   usp @ sp! texi ;
(S --)
: sched   sys_stks up@ tsem_clr robin?
  begin dup tsem_clr link @ until
  do_task ;
```

```
Scr # 14
\ user functions                                       has062991
: save_state   r>  r> up@ uip !  sp@ dws + up@ usp !
   rp@ up@ urp ! >r    ;
(S rcvr_addr--flag)
: ?refused   rstate @ refus and ;
(S rcvr_addr--last_msg_link_addr)
: find_end   rlist begin dup @ ?dup while nip mlink repeat ;
```

```
Scr # 15
\ registration                                         has063091
(S stack_size,words pgm_size,w #rcvr--size,words)
: tcb_size   0 swap r#>ra swap ws * + swap ws * +
  [ 70 ws * 80 + ] literal + ;
(S stack_size pgm_size #rcvr top_addr--bot_addr)
: top>bot   >r tcb_size r> swap - ;
(S up &string--)
: get_tname   tname 8 cmove ;
(S up--) \ interactive version for testing
\ : get_tname  tname dup 8 blank bl word count swap -rot cmove
```

2. If you are on a project that is new and interesting, and you are interested in letting others know and can write about the technical parts, we will find a writer that can reshape it into an article that will complement the technical aspects with general-interest aspects of Forth and will help find an appropriate place to get it published.

3. As articles are published, we will coordinate Forth- and FIG-related advertisement to be placed in those issues along with the articles.

Why are we emphasizing mid-level managers? They are the people who specify or can direct the technical details of a project. They are the people who put together a project team. They are the people who would most likely pale at new innovation without justification. They are the people who make a project succeed or fail. They are the people we have failed to reach. Who of us would complain if Forth were required in your project?

I am always available for comments.

—John Hall
415-535-1294
JDHALL on GEnie

(Letters, from page 16.)

CREATE, CREATE DOES>, and, in some Forths, also VARIABLE, will need to be recoded. As of today, if you are using Forth and you want your application to be compatible with future hardware and system software, you have to refrain from using some very natural Forth mechanisms. This is all very un-Forth-like!

Forth could and should, if only some more people bothered, evolve on the Macintosh. There is no reason why the vast programming utilities of MPW, its many tools, its great multi-scrollbar editor, could not become available to the Forth environment. I myself am depending more and more on the MPW environment and, yet, it still lacks the one thing taken for granted in any Forth environment: interactivity—the kind you get by being able to compile small entities and immediately execute and debug them.

I think that if Forth has proved to be a disappointment on the Mac, one has only oneself to blame. With Forth's simple and open "architecture," the compiler is not the limit, the user is.

Conrad Weyns
Bjerkebakken 62D
0756 Oslo 7
Norway

```
Scr # 16
\                                                          has062991
(S stack_size pgm_size #rcvr up--next_up)
: set_stacks   dup >r swap
  r#>ra dup r> swap >r >r  + 80 + tuck + 140 + dup r@ - 1+ r@
  swap erase tuck dup r@ urp0 ! r@ urp ! r@ splim ! 100 - dup
   r@ rplim ! 20 - dup r@ usp !
   r@ usp0 ! r> _dp + r> swap ! ;

Scr # 17
\ Link in a new task                                       has111390
(S up0@--&predecessor)
: find_predecessor   dup >r begin dup link @ dup r@ <> while
   nip repeat r> 2drop ;
(S &pred up0 &new &link--&pred up0 &new &link link flag)
: pri<?   2dup link @ dup priority @ rot priority @ u< ;
(S &pred up0 &new &link link--)
: do_links   rot tuck link ! swap dup >r link !
@ swap dup r> = if 2drop else link ! then ;
(S &new up0 up0@--)
: link_next   >r dup r@ find_predecessor swap
  2swap pri<? not if  begin nip   pri<?   over r@ = or
  until  then  r> drop do_links ;

Scr # 18
\ New task linking cont'd.                                 has111390
(S &new up0--)
: link_task   dup @ ?dup if link_next else over dup
  link !  ! then ;
\ Tcb linking test words
(S  priority--)
: doit here swap , 0 , up0 link_task ;
(S --)
: sl   up0 @ dup >r begin cr dup u. link @
  dup dup u. r@ = key? or until r> 2drop ;
```

U.K. Contest Winners

Dear Marlin,

I have enclosed two of the winning entries in the Forth Programmer competition *[see "Letters," last issue—Ed.]*, along with some comments from the entrants.

```
Scr # 19
\                                                          has062991

(S parm st_ip pri maxt up--)
: set_stuff   >r r@ maxtim ! r@ priority ! r@ uip !
  clock 2@ r@ actime 2!  suspbt r@ tstate !  r@ spmin on
   r@ rpmin on r@ tpush  r@ get_tname  'next unp !  ;

(S top_adr &tname parm st_ip pri maxt stksz pgm_size #rcvr id
   --top_of_next_tcb up)
: regis   idlock 8 roll over >r
  >r 3dup r> top>bot dup >r set_stacks r@ set_stuff
  r@ up0 link_task r> dup r> swap nrcvr ! idunlock dup ws -
  swap ;
```

```
Scr # 20
\                                                          has062991

(S rcvr_addr call_frame_addr--)
: m_link   mlink dup off over ?rsem over di find_end !
  rsem_clr ei ;

(S call_frame_addr--)
: set_ctime    cdelay @ dup 0>= if 0 clock 2@ d+
  up@ actime 2! [ delbt endobt or ] literal else
  drop endobt then up@ tstate set ;

(S call_frame_addr--)
: set_sender   up@ swap sender ! ;

(S #parms--) \ pop in current task
: trash_parms   ws * up@ usp +! ;
```

```
Scr # 21
\                                                          has062991
: setparms (S --)    clock 2@ 2dup start_time 2@ d-
  up@ totime d+! start_time 2!
  sp@ up@ splim @ u<  rp@ up@ rplim @ u< or
  sp@ up@ usp0 @ u> rp@ up@ urp0 @ u> or or
  stkbt and up@ tstate set
  sp@ up@ spmin @ umin up@ spmin !
  rp@ up@ rpmin @ umin up@ rpmin ! ;

 : enter (S --) r> save_state >r setparms ;

(S [data..] delay entry# dest_task--[new_data..] ret_parameter)
: call enter 3 trash_parms  sp@ [ msize csize - ]
  literal - dup >r sp!  r@ cdest @ r@ centry @ r#>ra dup
  ?refused if drop  #refused up@ tpush else
  r@ m_link r@ set_ctime r@ set_sender then r> drop sched ;
```

As an update to the report I sent, I would like to confirm that the discussions with IBM were successful, and they will be hosting our next London meeting. Incidentally, they have been most generous in supplying not only space in their South Bank Lecture Theatre, but also a buffet meal. Perhaps other FIG Chapters would be interested to know that there is some benefit to be made from approaching large corporations with regards to meetings. We are currently trying to encourage them to attend euroFORML.

The FANSI project is proceeding apace, and initial circuit diagrams have been drawn up for our planned processor board. We have actually settled on a 6309 processor, which is a CMOS 6809. This was chosen for its low cost and availability, and for the orthogonality of its instruction set and direct support for two stacks. It is possible that, in the future, we will produce a second board using a stack processor, very possibly the German device designed by Klaus Schleisiek-Kern, incidentally the organizer of this year's euroFORML.

Yours sincerely,

Gordon Charlton
Events Secretary, FIG-UK
31 Pikestone Close
Hayes, Middlesex UB4 9QT
Great Britain

* * *

I have been using Forth since 1978. Most of the professional work I have done has been embedded software in traffic monitoring equipment for an outfit called

Golden River. More recently, I wrote an implementation of Forth-83 for the Transputer. I started working on the Matching Birthdays problem as a result of Gil's blatant provocation in asserting it was a formidable calculation. My first attempt performed the entire calculation on the stack and ran to six screens of source code. My general thoughts about Forth are that it is a good thing.

Big Number Arithmetic and the Probability of Matching Birthdays

In the editorial of *Forth-write* 56, Gil gave the formula for calculating the probability of finding at least one pair of matching birthdays in a group of n people as $(1 - P)$, where P is found by multiplying all the integers in the range $(366 - n)$ to 365 inclusive, then dividing by 365 n times.

To obtain four decimal places using integer arithmetic, we must multiply by 10,000 and subtract the result from 10,000. All the multiplication must be performed before the division, giving rise to very big intermediate values. It is only necessary to multiply and divide by single-length numbers, so the algorithms for "short multiplication" and "short division" are used; the product or quotient is written, cell-by-cell, back to the same memory locations that held the corresponding cells of the multiplicand or dividend.

It is interesting to compare the results of this program with those which Gil obtained using the Monte Carlo method. The big num-

```
Scr # 22
\                                                          has111590
(S up--)
: enable   dup >r begin dup tstate suspbt swap reset
  link @ dup r@ = until r> 2drop ;
\S *****
(S --)
: startup   id " MASTER" drop 1- find 0= abort" No Master!"
  255 0 30 0 2 first_tcb_top
  regis MASTER
  up@ tpush up@ enable sched ;

Scr # 23
\                                                          has112090
(S receiver_specifier 'set or 'reset bit--)
: sr_all_rcvr  rot up@ 0 rstate rbnds rot drop do
  3dup 1 and * i rot execute u2/ rcvrsz +loop 2drop drop ;

Scr # 24
\ accept                                                   has112090
(S delay_parameter 0|selbt --)
: setdel   swap dup  if  dup 0< if >r up@ actime 2@ r> -1 d-
  else 0 clock 2@ d+ then  up@ actime 2! delbt then
  or up@ tstate set ;
(S rcvr_specifier--)
: open_rcvrs   ['] set open sr_all_rcvr ;
(S receiver_specifier delay_specifier -- &sender &data rcvr#)
: select
  enter 2 trash_parms  selbt setdel open_rcvrs sched ;
(S delay_specifier -- Dclock)
: delay   enter 1 trash_parms 0 setdel sched ;
```

```
Scr # 25
\ functions                                         has063091

(S sender--flag) \ true if msg not from interrupt (ixmit)
: ?~ix_msg   tstate @ [ delbt not ] literal <> ;

(S sender--) \ ADA end of critical section
: release   enter 1 trash_parms   dup ?~ix_msg if #call_ok
   over tpush then tstate endobt swap reset sched ;

(S rcvr_specifier--)
: busy
  enter 1 trash_parms ['] set refus sr_all_rcvr sched ;

: unbusy
  enter 1 trash_parms ['] reset refus sr_all_rcvr sched ;



Scr # 26
\ Ixmit                                             has063091

(S entry_# dest_task &msg--status) \ send msg frm interrupt
: ixmit    >r 2dup swap r#>ra dup ?refused
  if r> 2drop 2drop #refused
  else   rot r@ dest_rcvr ! ( dest_rcvr)
  swap r@ dest ! ( dest)  r@
  [ 0 tstate 0 mlink - ] literal - r@
  sender ! ( pseudo-sender)   r> m_link #call_ok then ;

\S  Message structure: link, dest, dest_rcvr#,
pseudo_sender, data...  mlink is set
to -1 and ~delbt (fffb) when rec'd. When released
endobt is cleared ( fff9).
The sender field is set so that the
mlink field appears as tstat


Scr # 27
\ break                                             has062991
(S up--up f)
: ?breakable   dup dup usp @ mdata - mlink @ 1+ swap
  tstate @  endobt delbt logic and 0<> ;

(S up|false--up'|false)
: >dest
  dup if ?breakable swap usp @ mdata - dest @ and then ;
(S up-- up'|false)
: bchase   dup dup >r begin over >dest 2dup <> dup
  if drop rot drop false -rot over r> link @ dup >r <>
  over and then   while rot drop swap
  repeat 2drop r> drop ;

: ppad (S --addr)   _dp @ 80 + ;
```

ber arithmetic routines may prove useful in other applications. *[See Figure One on page 7.]*

—Philip Preston

One-Screen Full-Screen Editor

Our company, M.A.S.S., operates as a bespoke software house working with a wide variety of projects, 90% of which are now Forth based.

Between jobs, I get involved in what can loosely be called R&D. This can range from developing software tools (tinkering with Forth), looking at ways to improve efficiency (making a case for the latest hardware), and exploring software techniques (playing games).

Seriously, though, we started to move over to Forth in about 1985, after studying the language (during R&D) and comparing it to the languages used at the time, i.e., BASIC, Pascal, and assembler. The benefits gained came slowly, at first, but then at an accelerating rate as experience was gained and the various "pennies dropped."

We tried several commercial versions of Forth, with varying levels of success, before developing homegrown versions. This gives us absolute control to customize as required. Forth makes you greedy!

One of the many benefits Forth gives us is the ability to very quickly produce prototypes. While our competitors are busy drawing up impressive-looking flowcharts and "tecspecs," we are demonstrating some sort of prototype to which the client usually can relate. This has worked very well for us. (As a point of interest, we have had over 12,000 Forth applications distributed world wide.)

Another R&D session looked long and hard at C. I obviously missed something. All that syntax!

Yet another period of R&D produced the one-page full-screen editor. We were working on text manipulation at the time, so I lifted some of the code and worked on minimizing it.

I would not have believed that any sort of screen editor would have fit into 1K of source code—"packed jumble" or not—especially as the editing facilities offered are quite respectable. The average words per definition worked out to fewer than six, and only an improvement to the word TYPE makes any significant improvement to performance.

I found the competition a worthwhile exercise and, indeed, have imposed the one-screen restriction on several pieces of code since. This approach forces a re-think and a re-work of just about every word used. Perhaps Gordon Charlton has unwittingly opened up a whole new programming technique? I cannot, however, recommend the regular use of the style used in my example! *[See Figure Two on page 36.]*

—Mike Lake

PDE Erratum

Dear Mr. Ouverson,

In my article on PDE screen management ("Add and Delete Screens in PDE,"

(Continued on page 37.)

```
Scr # 28
\                                                           has063091
\ Break a task deadlock
(S up_to_resume--flag, true if broken)
: break   dup bchase ?dup if
  over cancel_msg
  usp @ [ 0 dest_rcvr mdata - ] literal + @ over tpush
  #broken over tpush
  tstate endobt swap reset true else drop false then ;

(S --)
: dedlok   up0 @ dup >r begin dup break drop link @ dup
  r@ = until r> 2drop ;
```

(Editorial, from page 4.)

- Forth (50%)
  - Program design
  - Stack operations
  - Postfix notation
  - Forth compilation (definitions, conditionals, loops)
  - Number types and operations
  - Constants, variables, arrays
  - Dictionary structures
  - I/O operations
  - Control structures

3. Forth Programming (20%)

4. Forth and Professional English (10%)

Conscientious Computing

Omni magazine's May 1991 issue contains a guide to energy-efficient behavior, something many technophiles appreciate as a kind of self-preservation. Among their suggestions related to office work: use machines with controls that keep them from running at full power during work lulls; save your laser printers for output that demands their quality output (they require ten times as much energy as other printers); use a fax *without* a thermal print mechanism; use laptop computers, which are more efficient than desktop models; use smaller screens; use recycled paper; and carpool or, better yet (take my word for it), telecommute.

For quantity reprints of the complete *Omni Energy Efficiency Guide*, write to *Omni* Energy Guide, 1965 Broadway, New York, NY 10023.

—Marlin Ouverson
Editor

ATTENTION FORTH AUTHORS!

AUTHOR RECOGNITION PROGRAM

TO RECOGNIZE AND REWARD AUTHORS OF FORTH-RELATED ARTICLES, THE FORTH INTEREST GROUP HAS ADOPTED THE AUTHOR RECOGNITION PROGRAM.

Articles

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

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

Letters to the Editor

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

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

Figure Two. One-screen full-screen editor.

```
=====================================================================
  ONE SCREEN FULL SCREEN EDITOR                        Mike Lake
=====================================================================
    0 VARIABLE KT 512 ALLOT : Y R# @ C/L / ; : X R# @ C/L MOD ;
  : RANGE OVER + DUP 0< OVER 1023 > OR 0= IF SWAP THEN DROP ;
  : MOV R# @ SWAP RANGE R# ! ; : LEFT -1 MOV ; : RIGHT 1 MOV ;
  : UP -64 MOV ; : DOWN C/L MOV ; : HOME  0 R# ! ; : >E C/L X - ;
  : L0 SCR @ (LINE) ; : AD 0 L0 DROP R# @ + ; : LAD Y L0 ;
  : Y.L 0 OVER AT L0 TYPE ; : .L Y Y.L ; : CEOL AD >E BLANKS .L ;
  : .ED 16 0 DO I Y.L LOOP CR SCR ? ; : CRLF X MINUS R# +! DOWN ;
  : CL LAD BLANKS .L ; : >S UPDATE SCR +! .ED ; : N 1 >S ;
  : PUT LAD PAD SWAP CMOVE ; : !SCR HOME AD 1024 BLANKS .ED ;
  : GET PAD LAD CMOVE .L ; : TOGG KT 1 TOGGLE ; : P -1 >S ;
  : CLR BL AD >E 1- 2DUP OVER 1+ ROT ROT CMOVE + C! .L ;
  : !K DUP AD KT @ IF DUP DUP 1+ >E 1- -CMOVE C! .L ELSE C! DUP
    EMIT THEN RIGHT ; : DEL LEFT CLR ; : UNDO EMPTY-BUFFERS .ED ;
  : SET CFA SWAP 2* KT + 2+ ! ; : SETS 256 0 DO I ' !K SET LOOP ;
  : GO BEGIN X Y AT KEY DUP 2* KT + 2+ @ EXECUTE 0< UNTIL ; SETS
  : ED FLUSH SCR ! HOME .ED GO UPDATE ; : TAB 8 X OVER MOD - MOV ;
=====================================================================
  CORE WORDS
=====================================================================
  KT       ( ...ADDR)      FIRST 2 BYTES=FLAG 256*2 BYTES FOR KEY ACTIONS
  X        ( ...N)         X POSITION OF CURSOR
  Y        ( ...N)         Y POSITION OF CURSOR
  RANGE    ( CURSOR  N ...CURSOR)  LEAVES CURSOR+N OR ORIGINAL CURSOR POS
  MOV      ( N ...)        MOVES CURSOR BY N
  >E       ( ...N)         CHARACTERS FROM CURSOR TO END OF LINE
  L0       ( N...ADDR C/L) ADDRESS OF START OF EDITING LINE N
  AD       ( ...ADDR)      ADDRESS OF CURSOR
  LAD      ( ...ADDR C/L)  ADDRESS OF START OF CURRENT LINE
  Y.L      ( N...)         LOCATE AND PRINT LINE N
  .L       ( ...)          PRINT CURRENT LINE
  .ED      ( ...)          PRINT EDITING SCREEN
  >S       ( N ...)        MOVE ON N SCREENS
  GO       ( ...)          MAIN KEY LOOP ;
  ED       ( N ...)        INVOKES EDITOR USING SCREEN N
  SET      ( N PFA ...)    SET KEY N TO PERFORM WORD ACTION
                           USE AS IN...      13 ' CRLF SET
  SETS     ( ...)          SETS ALL KEYS TO DEFAULT.. (STORE KEY PRESS)
=====================================================================
  WORDS TO ASSIGN TO REQUIRED KEY PRESS
=====================================================================
  CLR      ************************* = CLEAR CHARACTER UNDER CURSOR
  DEL      * ALL THESE WORDS ARE    * = CLEAR CHARACTER TO LEFT OF CURSOR
  CEOL     * ASSIGNED TO KEYPRESSES.* = CLEAR FROM CURSOR TO END OF LINE
  CL       * ADDING EXTRA ACTIONS   * = CLEAR LINE
  !SCR     * IS A DODDLE. A NEW     * = CLEAR EDITING SCREEN
  UNDO     * ACTION HAS THE KEY     * = RESTORE EDITING SCREEN TO ORIGINAL
  N        * AVAILABLE ON THE STACK * = NEXT SCREEN
  P        * AND MUST LEAVE A VALUE * = PREVIOUS SCREEN
  PUT      * ON THE STACK. A MINUS  * = PUT CURRENT LINE TO PAD
  GET      * VALUE SIGNALS TO       * = GET CURRENT LINE FROM PAD
  TOGG     * EXIT THE EDITOR.       * = TOGGLE INSERT/OVERWRITE MODES
  !K       * *********************** = STORE KEY PRESS (DEFAULT ACTION)
  MINUS                               = FORTH WORD USED TO EXIT EDITOR
  LEFT RIGHT UP DOWN HOME CRLF TAB    = MOVE CURSOR
=====================================================================
  INSTRUCTIONS .. THE ABOVE EXPECTS THE FOLLOWING COMMON FORTH WORDS.
    -CMOVE ( ADD1 ADD2 COUNT ...) AS CMOVE BUT FROM HIGH TO LOW MEMORY.
    AT     ( X Y ...) LOCATE CURSOR AT X Y.
  LOAD THE SCREEN AND ASSIGN ACTIONS TO THE REQUIRED KEY PRESSES.
  13 ' CRLF SET      8 ' TAB SET     27 ' MINUS SET ETC.
  THE CONFINES OF ONE SCREEN RESULTS IN THE PACKED JUMBLE ABOVE
  HOWEVER DEFINITIONS ARE VERY SMALL AND EASY TO FOLLOW.
```

(Letters, from page 34.)

FD XIII/1), I supplied words that would enter the PDE editor automatically if a load error occurred—`(ERROR)`, `(WHERE)`, etc. on screen 11. At the time, I thought these words crashproof. I spoke too soon. Since then, I've discovered a rather obscure bug that will crash the system like a fireworks display.

This happens when you first load Forth from the CP/M or DOS command line and decide to provide as an argument a filename for Forth to open. Should you misspell the filename or give one that does not exist, an open-file error occurs, the PDE editor is entered, and crash.

Yet the same open error done from the Forth command line results in the expected error message and no problems.

The trouble starts back when you take the pristine Kernel83 and add the extensions and assembler. As a kindness, the load screen ends by doing the `SAVE-SYSTEM` for you. Because of this, the variable `BLK` is saved as having a value of one, as this is the load screen number.

When you first load in Forth, I found that the boot routine is provided by `START`, which in turn calls `DEFAULT`. `DEFAULT` searches CP/M's DMA (on my computer) for a filename to open. Normally, this happens smoothly, and the sequence exits through `QUIT`. But in an open file error, it exits through `ERROR`. Unfortunately, nowhere is `BLK` reset to zero beforehand. Now `(ERROR)` in my PDE uses a non-zero `BLK` as a flag that the error occurred during a load, so it enters the editor. This causes the system to crash.

However, in setting up the Forth command line, `QUIT` early on resets `BLK` to zero, so an error there displays the error message normally.

You can solve the problem at several levels.

1. Simplest is to redo the `SAVE-SYSTEM` from the Forth command line. This ensures that `BLK` is zero.
2. You can rewrite `START` to include `0 BLK !` before calling `DEFAULT`. It's easy to patch in, as `BOOT` is a deferred word. Then, future additions to the program using this load screen to save the system won't cause problems.
3. Ultimately, correcting `DEFAULT` to reset `BLK` and recompiling would guarantee that this gremlin won't return to haunt you.

Yours truly,
Walter J. Rottenkolber
P.O. Box 936
Visalia, California 93279

$ Contest Announcement $

Call for Papers!

Forth Dimensions is sponsoring a contest to encourage authors of articles about Forth and **"Object-Oriented Programming"**

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

Deadline: September 16, 1991
See editorial in issue XIII/1 for details!

Answer to a Dream

Dear Sir,

In answer to John G. Derrickson's letter requesting a low-cost, better Forth ("Dreaming That It's Forth," *FD* XIII/2), I would like to recommend Upper Deck Forth, available from Upper Deck Systems (P.O. Box 263342, Escondido, California 92026).

I have found it to be exceptionally fast and easy to use. It is a text-based system and incorporates such features as mouse control for the full-screen editor, direct access to DOS functions from the terminal input stream, full memory usage, disassembler, assembler, and support for generating headerless, turnkey applications.

The only difficulty I encountered when using Upper Deck was while writing an assembler interrupt routine that required absolute addressing. The dynamic memory allocation readjusted segment and offset addresses during compilation, so determination of the exact location of the interrupt routine was made more difficult. This was solved by readjusting the code within the interrupt routine after compiling.

I highly recommend this MS-DOS Forth system.

Sincerely,
Glen F. Ingle
1585 Samedra Street
Sunnyvale, California 94087

I would not have believed that any sort of screen editor would have fit into 1K of source code.

News from the on-line Forth RoundTable

Best of GEnie

In the February column I promised not to delay so long between recaps of guest conferences. Before making good on that promise, I want to call your attention to the improved appearance of the GEnie Forth RoundTable. Topics and entire categories are much cleaner. The rat's nests of messages, many of which my ForthNet message ports created, are largely gone. All this can be attributed to our newest SysOp, Elliott Chapin. Elliott came on board knowing he would have to pick up after the likes of me, so he is either one heck of a good guy or a glutton for thankless jobs. I happen to believe the "nice guy" theory will stand the test of time.

Here, in his own words, is Elliott's introduction: "I was born in 1942 in New York City, just five days before my better known half-nephew Harry. A couple of years later, my father James Chapin, an established artist, moved the family out to rural northwest New Jersey. Now I have a family of my own in Toronto. I studied math at Princeton (starting my acquaintance with computers there) and Columbia in the 60s. Since then, I have found various kinds of work in the arts and education, but I also drive a taxi when necessary. I am enjoying the learning opportunities available on the GEnie's Forth RoundTable, while doing "slash and stash" messagebase edits as the latest assistant SysOp." (Note: ELLIOTT.C is Elliott's e-mail address.)

Welcome to the funhouse, Elliott.

* * *

It is true that revisits with our guests in this column never capture the intimacy of the Real-Time Conference where you pose the question, usually sparked by another comment, but they do serve to remind attendees of poignant exchanges and to let non-participants know what they missed. Transcripts of the complete conferences are archived in the GEnie Forth RoundTable's Software Library 1. Please make a note, as you scan this column, to download those you find of value, because the conferences as presented in this column are reduced to the guests' opening remarks.

Guest conferences reviewed here include Jef Raskin, "What Happened to the Cat?"; Alan Furman, "ACM SIGForth Update"; Bill Muench, "Embedded with eForth"; Guy Kelly, "Forth and Industry"; Dean Sanderson, "Addressing Management Concerns" (regarding use of Forth); Roy Martens and Glen Haydon, "MVP is Alive and, well..."; Charles Johnsen, "Mutable Instruction Set Computers."

Jef Raskin, creator of the Mac and Canon Cat, asked "What happened to the Cat?" in reference to the marketing failure of the Canon Cat. This initial discussion rapidly gave way to a more thorough look at what the interface between computer and man should be—certainly an area, it can be safely argued, in which Jef Raskin is one of the ranking authorities, if not the authority. 10/17/90

Jef Raskin: I am pleased to be here, and I think that it might be best to just start with questions. When I see the directions the questions take, I can make more extended comments.

Cool CAT JAX: First I were a Programmer... Now I am a Project Engineer... Jef, how long do I have to wait before I start having Ideas? :-)

Jef Raskin: Thank you Mr. Cat. My real interest these days is in interfaces. For example, windows are dumb, icons wrong, and mouses a nuisance.

Dennis Ruffer: Sticking to that "heresy," let me ask how/why did you start the Mac project?

Jef Raskin: I can't stand the usual hand-to-mouse existence. The real question is why we have to bother with all those "features" when we are trying to get something done. I started the Mac project in 1979. I have learned something in the intervening decade. In those days I was (correctly, I think) inspired by the work at PARC. But one mustn't confuse "better" with "good." The Mac was better, but one can go a lot better.

Wil Baden: Please tell us what you think happened to the CAT.

Jef Raskin: It was, I have been told, a victim of internecine warfare within Canon and a lack of marketing support. I am sure of the second hypothesis.

Gary Smith: There was excitement here after John Bumgarner was in conference with Steve Roberts about the possibility of a portable, personal Forth laptop then *bingo!* the CAT is on sale in Service Merchandise. That's a lot of smoke and mirrors. Any further comment?

Jef Raskin: They are, by the way, still available. A guy, David Wing, in San Diego has kept track of them. Canon did not know what it had, and tried to sell it through its electronic typewriter division. This was a mistake. Then they tried selling it for twice the design price. That was a mistake. We did make some prototype portables, I have one here and it works (and runs Forth), but our Vulture Capitalists had lost their stomachs (and minds).

Dr. Alan Furman, independent software and electronics consultant, presented an *update on a sister organization, SIGForth ACM. As a principle of this group's genesis, Dr. Furman was more than qualified to do so. 11/15/90*

I should begin by acknowledging the tremendous contribution of George Shaw, who has been the driving force in SIGForth from the beginning. My main role was as a crusader, in late 1988, for some kind of organization that would specialize in professional issues and the commercialization of Forth. FIG's constituency is a mixture of professionals and hobbyists. And many of those professionals didn't care who admired Forth; they were in a position of having discretion over programming language. Whenever you have a group of people together, you have to stick to things that they are all interested in at the same time, which in FIG's case is Forth technology and sharing of ideas.

I will close with some exciting news. The Forth community in Leningrad would like to put on a conference either Fall 1991 or Spring 1992, which will be run in cooperation with SIGForth.

Bill Muench, president of Ontologic and coauthor of eForth, discussed eForth, the .asm Forth kernel for the 90s. 12/13/90

eFORTH is derived from my commercial version bFORTH. Both are ANS Forth subsets. I designed eFORTH for ease of implementing new systems, to easily fit in 8K bytes, to have a minimum of machine code and some debugging tools. My debug tools are `TX!`, the transmit primitive, `.S`, `DUMP`, and `WORDS`. I use a simple metacompiler rather than MASM, which means I first write an assembler. That is how I get to know the processor.

After the new system is running, I optimize it, first loading the assembler, then new code words. When working, they are added to the kernel, repeat 'til done.

I use my host file server, bHOST, to upload source text files to the target system. MASM was chosen by Dr. C.H. Ting as a vehicle to communicate eFORTH. It is commonly available and on a widely used platform, the PC. But any assembler will do. Possible alternatives are spreadsheets or word processors with macros.

Guy Kelly expanded on his November 1990 FORML theme, "Forth in Industry." Guy has a vast array of Forth-driven industrial projects to his credit to support his views that Forth is an excellent platform for such endeavors. 1/17/91

Experiences as a Forth producer, user, and teacher convince me that it's almost impossible to get a professional programmer to use Forth and almost impossible to discourage a professional engineer from using Forth.

Now that Laxen, Perry, and Zimmer have handled the "high end" and Dr. Ting is attacking the "low end," how about the "Forth as a hardware development tool" area?

Dean Sanderson, software engineer with FORTH Inc., discussed "Addressing Management Concerns over use of Forth as an Applications Platform." 2/21/91

Because of the power that Forth has given us (to keep projects small and quick), we have been able to avoid learning what others have had to about software development. Those who have grown up with Forth do management by intuition. We have trouble communicating with those who've been successful using Fortran, C, or assembler. It's as if we speak different languages. As projects escalate, we find we have not killed the dragon, only maimed him. As we ready for battle, we find our pride has left us with few new weapons.

For Forth to survive as a respected language, it must prove its adaptability and change enough to support the concerns of management. These include: Integration, Maintenance, Documentation, Declining cost, Q.A., Configuration, and Scheduling.

Though we've started late, we can survive by capitalizing on what others have learned.

This conference's invited co-guests were Roy Martens, president of Mountain View Press, with Glen Haydon, author of MVP Forth. Glen discussed the fate of MVP in the 1990s market. 3/4/91

Thank you for inviting Mountain View Press. We are alive, but changes are in store. Roy has sold his home and is moving to San Francisco. He is taking an apartment April 1. We have agreed for me to take over Mountain View Press. As some of you know, Phil Koopman, has taken a new job which requires him to divest himself of WISC Technologies. Epsilon Lyra is my company. I am bringing WISC Technologies in as a division. Roy and I agree to doing the same with Mountain View Press.

I plan on carrying the public-domain versions of Forth which I included in my new edition of *All About Forth*: fig-FORTH, MVP-FORTH, F83, and F-PC. I will also carry available documentation.

We will also carry professional implementations from vendors who will make appropriate distributor's agreements. For the present, we will keep the same mailing address and phone numbers. We look forward to continuing our support of Forth users.

Charles Johnsen, President of MISC, Inc. (Mutable Instruction Set Computer, formerly Minimum Instruction Set Computer) was our invited guest. Charles was joined by Dr. David Fox, MISC's Software Engineer as the two discussed this new silicon engine. 4/18/91

Thanks for the welcome. I wanted to speak about the Silicon Palimpsest this evening. That is the processor without a fixed instruction set. MISC originally stood for Minimum Instruction Set Computer. It was a tiny company (and still is), set up to create a Forth stack engine.

We wanted to do something different from Novix. We wanted a processor for embedded control, not desktop computing. That guided our efforts and our business. Today we have changed our name to Mutable Instruction Set Computer, Inc., because we have an even better idea. By using FPGA (field-programmable gate arrays), we believe we can create a processor with a mutable instruction set. The advantage of a mutable instruction set is that custom instructions can be designed for improved performance.

—Gary Smith
GARY-S on GEnie

reSource Listings

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

Forth Interest Group

The Forth Interest Group serves both expert and novice members with its network of chapters, *Forth Dimensions*, mail-order services, and on-line activities. For membership information, or to reserve advertising space, contact the administrative offices:

Forth Interest Group
P.O. Box 8231
San Jose, California 95155
408-277-0668
Fax: 408-286-8988

Board of Directors	*Founding Directors*
John Hall, President	William Ragsdale
C.H. Ting, Vice-President	Kim Harris
Mike Elola, Secretary	Dave Boulton
Dennis Ruffer, Treasurer	Dave Kilbridge
Wil Baden	
Jack Brown	
David Petty	
Dennis Ruffer	

In Recognition

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

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

ANS Forth

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

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

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

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

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

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

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

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

Forth Instruction

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

On-Line Resources

To communicate with these systems, set your modem and communication software to 300/1200/2400 baud with eight bits, no parity, and one stop bit, unless noted otherwise. GEnie requires local echo.

GEnie
For information,
call 800-638-9636

- Forth RoundTable *(ForthNet*)*
  Call GEnie local node, then
  type M710 or FORTH
  SysOps:
  Dennis Ruffer (D.RUFFER),
  Scott Squires (S.W.SQUIRES),
  Leonard Morgenstern (NMORGENSTERN),
  Gary Smith (GARY-S)

- MACH2 RoundTable
  Type M450 or MACH2
  Palo Alto Shipping Company
  SysOp:
  Waymen Askey (D.MILEY)

BIX (ByteNet)
For information,
call 800-227-2983

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

- LMI Conference
  Type LMI at the : prompt
  LMI products
  Host:
  Ray Duncan (RDUNCAN)

CompuServe
For information,
call 800-848-8990

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

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

Unix BBS's with forth.conf (ForthNet and reachable via StarLink node 9533 on TymNet and PC-Pursuit node casfa on TeleNet.)*

- WELL Forth conference
  Access WELL via CompuserveNet
  or 415-332-6106
  Fairwitness:
  Jack Woehr (jax)

PC Board BBS's devoted to Forth (ForthNet)*

- British Columbia Forth Board
  604-434-5886
  SysOp: Jack Brown

- Grapevine
  501-753-8121 to register
  501-753-6859
  StarLink node 9858
  SysOp: Jim Wenzel

- Real-Time Control Forth Board
  303-278-0364
  StarLink node 2584 on TymNet
  PC-Pursuit node coden on TeleNet
  SysOp: Jack Woehr

Other Forth-specific BBS's

- Laboratory Microsystems, Inc.
  213-306-3530
  StarLink node 9184 on TymNet
  PC-Pursuit node calan on TeleNet
  SysOp: Ray Duncan

- Knowledge-Based Systems
  Supports Fifth
  409-696-7055

- Druma Forth Board
  512-323-2402
  StarLink node 1306 on TymNet
  SysOps: S. Suresh, James Martin, Anne Moore

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

- DataBit
  Alexandria, VA
  703-719-9648
  PCPursuit node dcwas
  StarLink node 2262
  SysOp: Ken Flower

International Forth BBS's

- Melbourne FIG Chapter
  (03) 809-1787 in Australia
  61-3-809-1787 international
  SysOp: Lance Collins

- Forth BBS JEDI
  Paris, France
  33 36 43 15 15
  7 data bits, 1 stop, even parity

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

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

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

- NEXUS Servicios de Informacion, S. L.
  Travesera de Dalt, 104-106, Entlo. 4–5
  08024 Barcelona, Spain
  + 34 3 2103355 (voice)
  + 34 3 2147262 (modem)
  SysOps: Jesus Consuegra, Juanma Barranquero
  barran@nexus.nsi.es (preferred)
  barran@nsi.es
  barran (on BIX)

**ForthNet is a virtual Forth network that links designated message bases in an attempt to provide greater information distribution to the Forth users served. It is provided courtesy of the SysOps of its various links.*

This list was accurate as of August 1991. If you know another on-line Forth resource, please let me know so it can be included in this list. I can be reached in the following ways:

Gary Smith
P. O. Drawer 7680
Little Rock, Arkansas 72217
Telephone: 501-227-7817
Fax (group 3): 501-228-9374
GEnie (co-SysOp, Forth RT and Unix RT): GARY-S
Usenet domain.: uunet!ddi1!lrark!glsrk!gars

FIG Chapters

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

Forth Interest Group
P.O. Box 8231
San Jose, California 95155

U.S.A.

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

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

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

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

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

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

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

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

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

  Stockton Chapter
  Doug Dillon (209) 931-2448

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

INTERNATIONAL

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

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

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

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

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

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

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

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

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

- **GERMANY**
  Germany FIG Chapter
  Heinz Schnitter
  Forth-Gesellschaft e.V.
  Postfach 1110
  D-8044 Unterschleissheim
  (49) (89) 317 3784
  e-mail uucp:
  secretary@forthev.UUCP
  Internet:
  secretary@Admin.FORTH-eV.de

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

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

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

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

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

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

SPECIAL GROUPS

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

"IBM will be hosting the next London meeting of FIG-U.K."

SEE "LETTERS"

FORML CONFERENCE

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

Following Thanksgiving, November 29–December 1, 1991

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

Theme: Simulation and Robotics

Papers are invited that address relevant issues in the development and use of Forth in simulation and robotics. Virtual realities, robotics, and graphical user interfaces are topics of particular interest. Papers about other Forth topics are also welcome.

Attendees are invited to enter a robot in a robotics contest where the robot solves a puzzle.

Mail abstract(s) of approximately 100 words by September 1, 1991 to **FORML Conference, Forth Interest Group, P.O. Box 8231, San Jose, CA 95155.**

Completed papers are due November 1, 1991.

Conference Registration

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

Conference attendee in double room—$350 • Non-conference guest in same room—$200 • Children under 17 years old in same room—$140 • Infants under 2 years old in same room—free • Conference attendee in single room—$450
Forth Interest Group members and their guests eligible for ten percent discount on registration fees.

Register by calling the Forth Interest Group business office at (408) 277-0668 or writing to: **FORML Conference, Forth Interest Group, P.O. Box 8231, San Jose, CA 95155.**

FORTH
INTEREST GROUP
1330 South Bascom Ave., Suite D
San Jose, CA 95128