$10 Volume XIII, Number 2 July 1991 August

FORTH DIMENSIONS

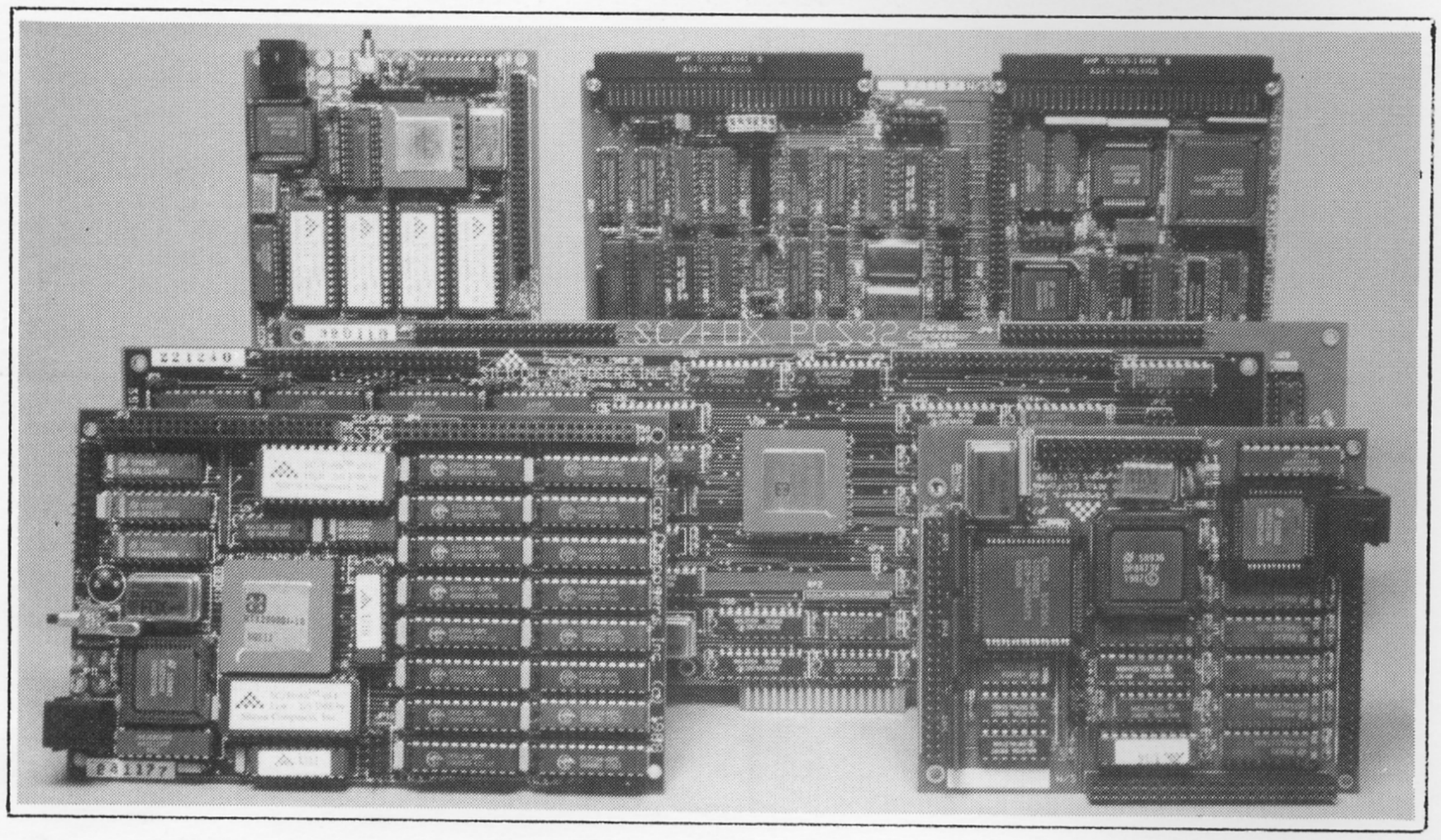
Neural Network Tools

Pygmy Forth

Smart Comments & Compiler Words

SILICON COMPOSERS INC

FAST Forth Native-Language Embedded Computers



Harris RTX 2000[tm] 16-bit Forth Chip

- 8 or 10 MHz operation and **15 MIPS** speed.
- 1-cycle 16 x 16 = 32-bit multiply.
- 1-cycle 14-prioritized interrupts.
- two 256-word stack memories.
- 8-channel I/O bus & 3 timer/counters.

SC/FOX PCS (Parallel Coprocessor System)

- RTX 2000 industrial PGA CPU; 8 & 10 MHz.
- System speed options: 8 or 10 MHz.
- 32 KB to 1 MB 0-wait-state static RAM.
- Full-length PC/XT/AT plug-in (6-layer) board.

SC/FOX VME SBC (Single Board Computer)

- RTX 2000 industrial PGA CPU; 8, 10, 12 MHz.
- Bus Master, System Controller, or Bus Slave.
- Up to 640 KB 0-wait-state static RAM.
- 233mm x 160mm 6U size (6-layer) board.

SC/FOX CUB (Single Board Computer)

- RTX 2000 PLCC or 2001A PLCC chip.
- System speed options: 8, 10, or 12 MHz.
- 32 KB to 256 KB 0-wait-state SRAM.
- 100mm by 100mm size (4-layer) board.

SC32[tm] 32-bit Forth Microprocessor

- 8 or 10 MHz operation and **15 MIPS** speed.
- 1-clock cycle instruction execution.
- Contiguous 16 GB data and 2 GB code space.
- Stack depths limited only by available memory.
- Bus request/bus grant lines with on-chip tristate.

SC/FOX SBC32 (Single Board Computer32)

- 32-bit SC32 industrial grade Forth PGA CPU.
- System speed options: 8 or 10 MHz.
- 32 KB to 512 KB 0-wait-state static RAM.
- 100mm by 160mm Eurocard size (4-layer) board.

SC/FOX PCS32 (Parallel Coprocessor System32)

- 32-bit SC32 industrial grade Forth PGA CPU.
- System speed options: 8 or 10 MHz.
- 64 KB to 1 MB 0-wait-state static RAM.
- Full-length PC/XT/AT plug-in (6-layer) board.

SC/FOX SBC (Single Board Computer)

- RTX 2000 industrial grade PGA CPU.
- System speed options: 8, 10, or 12 MHz.
- 32 KB to 512 KB 0-wait-state static RAM.
- 100mm by 160mm Eurocard size (4-layer) board.

For additional product information and OEM pricing, please contact us at:
SILICON COMPOSERS INC 208 California Avenue, Palo Alto, CA 94306 (415) 322-8763

Contents

Features

6 ***Smart Comments and Compiler Words***
William H. Stewart

If same program must run on different hardware configurations, a single code change may affect several versions of the program. "Smart comments" are a handy alternative to the tedious and error-prone alternative of manually editing multiple source files.

In Forth, new words are often developed to direct the compilation of a program in specific ways. But those words usually do not belong in the final application. Smart comments offer a way to remove such compile-time tools from the final code without affecting later dictionary searches.

9 ***Neural Network Words***
Tim Hendtlass

Neural networks are good at certain tasks the human brain is good at, like pattern recognition, often outperforming traditional computing techniques. A neural network is an interconnection of special processing elements that is based on the brain; each element has inputs and outputs, and is termed a "neurone." Neurones are arranged in logical layers, and a collection of such layers comprises a network.

The code in this article allows construction of a class of simulated networks, and their training by the widely used technique called "back propagation." It is portable to many environments, and other training techniques may be incorporated.

25 ***Introduction to Pygmy Forth***
Frank Sergeant

The author describes his Pygmy Forth as a fast, direct-threaded Forth for MS-/PC-DOS machines, complete with editor, assembler, and metacompiler. The entire system, including program, full source code, and documentation, can fit on a 360K floppy. Pygmy is intended to be a complete Forth for real work. It can metacompile itself, simply, so you can change anything you don't like.

Pygmy Forth is loosely based on Charles Moore's cmFORTH, and offers great educational opportunities as a small system that beginners or students can comprehend in its entirety in a relatively short time. Just how fast is it? Read the article and make your own comparisons...

Departments

4 ***Editorial*** ..... Transitions, article contest, implementation notes

5 ***Letters*** ..... Dreaming That it's Forth; IIe or Not IIe?; Implementation Infinitum; Membership Hostage.

24 ***Advertisers Index***

32 ***President's Letter*** ..... Organization, issues, actions, and explanations

33 ***Best of GEnie*** ..... Frank Sergeant, George Nicol, Bill Ragsdale, Glen Haydon, and Dick Miller

36–39 ***reSource Listings*** ..... FIG, ANS Forth, classes, on-line connections, FIG chapters

Editorial

Votes of Thanks

Some behind-the-scenes changes have taken place at *FD*, concurrent with the recent redesign of our inside pages. Thanks to smooth teamwork on the part of everyone involved, the transition happened seamlessly. I'd like to thank the two people who contributed so much to this magazine during their time with it—we appreciated them all along, but never more than since their responsibilities were integrated with those of the editor!

Kent Safford is an executive at the Association Development Center of San Jose, California, the company managing FIG's daily operations and tending to many of its business matters. Over the years, Kent has contributed greatly to the Forth Interest Group in convention planning, chapters organization, design matters, and many other areas. Beginning with *FD* issue VIII/3, he was given masthead credit as our Advertising Manager—though he also ably coordinated our printing and mailing—and he continued as our primary contact with advertisers, vendors, and service providers until this issue. His professional and highly organized approach made the job's intricacies look simpler than they are, and his congenial manner helped everyone to work together better. We are glad that FIG will still enjoy his influence via his work at ADC, although other duties have called him away from the *FD* position.

Cynthia Berglund worked as *FD*'s freelance graphic artist and single-handed production staff from issue VI/5 through the end of our last volume. For over six years, she was a dependable and talented member of our geographically dispersed publishing staff. We thank Cynthia for the years of service, for the late evenings dealing with errant file transfers á la modem, and for being a friendly voice in the face of looming deadlines and production schedules.

F-PC Author Needed

If we can learn from studying each other's Forth applications, how much more may be gained from familiarity with Forth implementations that differ significantly from our own cozy systems? In order to share some of the thematic variations among today's Forths, *FD* is publishing profiles of several Forth models and/or products. Our last issue featured eForth, and this time we shift focus to the lean and mean, with an introduction to Pygmy Forth. It would be natural to follow up soon with a discussion of the forward-looking and feature-rich F-PC (a.k.a. F-TZ). If you are familiar with that Forth and are willing to serve as tour guide/docent for readers who have not yet plumbed its depths, please write me at the FIG office or at MARLIN.O on GEnie. Do the same if you'd like to dissect some other Forth for our collective edification!

Object-Oriented Programming Contest

Don't forget our prize-paying contest for articles about Forth and object-oriented programming. The deadline is September 16, so dust off that keyboard and join the fray. For details, read last issue's editorial; or cut to the bottom line (i.e., the prizes) by turning to the ad elsewhere in this issue.

Of course, authors of Forth-related articles on other subjects, published in *FD* or elsewhere, still qualify for our usual author recognition program (see details on page 30).

Changes Elsewhere...

Mountain View Press, long-time publishers of

(Continued on page 30.)

Forth Dimensions
Volume XIII, Number 2
July 1991 August

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!

Dreaming That It's Forth

Dear Editor,

The "defector" (Letters, *FD* XIII/1) sheds light on the problem with Forth. We must thank Laughing Water by fixing the problem. Maybe we can win him back!

I want to program in Forth, but I use Quick BASIC. It costs only $70 and has an excellent interactive development environment. But it's BASIC and not Forth. I love Forth. While programming with QuickBASIC, I sometimes daydream that it's really Forth. A very pleasurable dream.

better than QuickBASIC in price and performance. *Forth Dimensions* can help. How about exhaustive descriptions and opinionated tests of the Forths in the market? Include shareware and freeware. Maybe when they've been criticized in public and been showered with suggestions, we'll have some Forths that compete with the BASICs, Cs, and Pascals in the marketplace. I believe one could win product-of-the-year awards by 1993.

Virtually yours,
John G. Derrickson

How about exhaustive descriptions and opinionated tests of the Forths in the market?

Every time I look at a new version of F-PC, I hope, "This is it. This is the Forth I've been waiting for." To be sure, I really appreciate what Tom Zimmer has done. It is so close to my dream Forth, I'm mad with expectations for his next release.

If HS/FORTH is what I've been waiting for, I'll never know. It's too expensive. And what of the other Forths? What we need is a Forth

IIe or Not IIe?

Note:

This will be my last year as a FIG-er unless you can show me a few more Apple IIe articles! I do not use either an IBM PC *or* a Mac. I can't find *one* ProDOS-based Forth disk or listing so I can update to ProDOS or F83! I'm *still* using my ancient Apple II MVP-FORTH! There *must* be someone out there (of the five million Apple IIe's bought) who uses Forth... There are some shareware Forths out there (in America On-Line), but they are all Apple IIGS-specific.

Keith Brewster
1152 Snowberry Ct.
Sunnyvale, California 94087

FD must limit the system-specific material it publishes—knowing that readers who learn from the code we print will usually find ways to implement it on their own systems—but we will gladly consider articles dealing with common or especially interesting details of Forth on less-than-mainstream computers.

Meanwhile, do any readers have advice for Keith?

—Ed.

Implementation Infinitum

Dear Sir:

I feel that a survey of FIG members—in which any member who has developed a dialect would submit a brief outline of the features and design goals of his implementation—would be fascinating. Comments on adaptations to best utilize the hardware or to provide extensibility are of most interest.

One of the drawbacks to widespread use of Forth is the ease of implementation of the language on a multitude of processors. It is this drawback that makes Forth so appealing to me. I have modified fig-Forth for the 8088 so that it is unrecognizable. It serves my purposes, but is not Forth-83 or fig-Forth standard.

On a different note, if the Forth Interest Group is facing economic challenges, shouldn't this be discussed and solutions sought out in the letters section of *Forth Dimensions*?

I have included a contribution to FIG for use in the Forth Reference Library, because of the quality and number of useful programming examples I have discovered there. The amount of examples available for Forth makes it an excellent learning language for anyone who is not living in a major center or working with a group of programmers.

Thanks,
Tom Saunders
Sigma 3 Engineering, Ltd.
300 Sigma Place
12120 - 106 Avenue
Edmonton, Alberta
Canada T5N 0Z2

Membership Hostage: Funds for Figures

To the FIG Board of Directors,

After five years as a member of FIG and contributor to *Forth Dimensions*, I have decided not to renew my membership.

My FIG renewal notice reads like the ones from magazines at the edge of bankruptcy. It says a $10 per year rate hike is necessary "in order to meet the rising cost of important

(Continued on page 30.)

Smart Comments and Compiler Words

William H. Stewart
Ben Franklin, Texas

In the real world, it is not unusual to have versions of the same program that must run on different hardware configurations. When this happens, a single change may involve editing several different program versions to make the same (hopefully) change. Additionally, we frequently create new words to help steer the compiling of a program. These words are necessary only during the compile mode and really do not belong in the final application EPROM. This discussion will present a method of using "smart comments" that permits using a single source program for all hardware applications, and a method of removing "compiler words" from the EPROM image. The illustrations are based on applications using the New Micros F68HC11 MPU.

I need opening comments that can be turned on or off, depending on the version of the program being compiled.

Smart Comments

The design approaches presented are not original. The idea for smart comments for F68HC11 applications was developed based upon programming concepts used in F-PC and on the need to provide programs for applications that had different hardware configurations.

The basic concept of smart comments is the F83 word (, the left parenthesis or opening comment. In the F68HC11 Max-Forth kernel, an opening comment may be terminated by a right parenthesis,), or by a carriage return. I needed a way to create opening comments that could be turned on or off, depending on the version of the program being compiled.

Consider the words in Figure One. \L and \S are our smart comments. Constants LARGE-KEYBOARD and SMALL-KEYBOARD are the flags that enable/disable the smart comments. To illustrate how they might be used, consider the programming example in Figure Two.

The described purpose of A? is to return a t/f flag, depending on whether the A key has been activated. The coding depicts two different keyboard/hardware configurations. The large keyboard has the A key mapped at address BD2F and the key appears as bit three, i.e., a value of eight. The small keyboard has the A key mapped at address BD03 and the key appears as bit two, i.e., a value of four. If the constant LARGE-KEYBOARD is made non-zero before the compile, and of course SMALL-KEYBOARD would be made a zero, then the word \L would execute as a no-op and the line of code BD2F C@ 8 AND 8 = would be compiled. Additionally, with SMALL-KEYBOARD at zero, the word \S would execute as the word (, by deleting the subsequent line of the program. By reversing the values of the flags LARGE-KEYBOARD and SMALL-KEYBOARD, the action would reverse and the statement BD03 C@ 4 AND 4 = would compile.

A family of smart comments may be implemented and nested. Without presenting the source program, consider the action of the words in Figure Three. Under the definition for a large keyboard, we have added the action of having two hardware versions: version A uses the \A to compile its source program lines and the no-change version uses \N to compile its source program lines.

The concept may be extended to whatever limits the programmer desires. However, a word of caution: since a closing comment,), will terminate a comment entry, this word must be avoided in lines of code that use a smart comment. Consider the action of the code shown in Figure Four-a; those lines of code will cause problems, since the 8 = and the 4 = will always be compiled. When comments are needed, place them at the end of the line and use only the opening comment (, as in Figure Four-b.

Compiler Words

Frequently, we need special words to aid in the compilation of a source program, but the use of these words ends at the termination of the compile. Typical compiler words are the smart comments described above and the New Micros ASM68 assembler; I also use a file called MC, that provides helper words for hand assembly coding, and a file called BINDER which is used to strip selected heads off words that are going into EPROM. I do not need or want any of these words to take up space in the application EPROM.

Compiler words may be deleted from the application code by shifting and relinking the dictionary.

When you re-map the F68HC11 for the compile, FORGET TASK makes the word (the last entry in the Max-Forth kernel. (The following sequence is illustrated in Figure Five.) Then move the dictionary pointer DP into an area of RAM that will not be used while compiling the application code. Now load all of your com-

(Continued on page 35.)

Figure One. Setting up basic smart comment words.

```
01 CONSTANT LARGE-KEYBOARD     ( set to 1 if true else 0

00 CONSTANT SMALL-KEYBOARD     ( set to 1 if true else 0

: NOP ;       ( does nothing

: \L          ( large keyboard smart comment
  LARGE-KEYBOARD
  IF
    NOP
  ELSE
    [COMPILE] (
  THEN
; IMMEDIATE

: \S          ( small keyboard smart comment
  IF
    NOP
  ELSE
    [COMPILE] (
  THEN
; IMMEDIATE
```

Figure Two. Example use of smart comments.

```
          ( --- t/f Returns a true/false flag, based on whether
          (         an A key has been activated.
          : A?
\L          BD2F C@ 8 AND 8 =
\S          BD03 C@ 4 AND 4 =
          ;
```

Figure Three. Nesting smart comments.

```
          : A?
\L \A       BD2F C@ 2 AND 2 =
\L \N       BD2F C@ 8 AND 8 =
\S          BD03 C@ 4 AND 4 =
          ;
```

Figure Four-a. Accidentally causing unconditional compilation.

```
          : A?
\L          BD2F C@ 8 AND ( A KEY) 8 =
\S          BD03 C@ 4 AND ( A KEY) 4 =
  ;
```

Figure Four-b. Correct placement of comments.

```
          : A?
\L          BD2F C@ 8 AND 8 =          ( A KEY
\S          BD03 C@ 4 AND 4 =          ( A KEY
          ;
```

CALL FOR PAPERS

for the thirteenth annual

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.

Presentations emphasizing virtual reality systems are being planned. 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, P.O. Box 8231, San Jose, CA 95155.

Completed papers are due November 1, 1991.

Registration and robotic contest information may be obtained by telephone request to the Forth Interest Group (408) 277-0668 or by writing to FORML, P.O. Box 8231, San Jose, CA 95155.

The Asilomar Conference Center combines excellent meeting and comfortable living accommodations with secluded forests on a Pacific Ocean beach. Registration includes use of conference facilities, deluxe rooms, all meals, and nightly wine and cheese parties.

This conference is sponsored by FORML, an activity of the Forth Interest Group. Information about membership in the Forth Interest Group may be obtained from the Forth Interest Group, P.O. Box 8231, San Jose, California 95155, telephone (408) 277-0668.

Neural Network Words

Tim Hendtlass
Hawthorn, Victoria, Australia

THE FIRST OF TWO PARTS

A neural network is an interconnection of special processing elements. Both the processing elements themselves and the method(s) used to interconnect them are based on our understanding of the way the human brain is constructed. Since the brain has millions of these processing elements (called neurones) and our networks only tens or hundreds, it is not reasonable to expect that we could mimic the full range and scope of human cognitive behaviour. However, neural networks can show an aptitude for certain tasks that the brain is good at—such as pattern recognition—often performing the task better than traditional computing techniques.

Despite being almost trivial, such a neurone can distinguish between two types of input.

The code developed in this article will allow the construction of a class of simulated networks (called non-linear feed-forward networks) and their training by a technique called "back propagation." This is the single most common architecture and training technique in use today. The code is written using the F-PC package produced by Tom Zimmer. With minor modifications, it is readily adaptable to other Forth dialects, other architectures, and other training algorithms.

Quick Introduction to Artificial Neural Networks

The basic building block, a neurone simulation, is shown in Figure One. It has a number of inputs, one (called the BIAS) always connected to +1 and the rest supplied with values by the user. Each input has a weight associated with it which controls how much effect that particular input will have on the output of this neurone. The weights are represented in Figure One by a circle with a W in it. If the weight is zero, the actual input value is irrelevant. Weights can be positive or negative, as can the input signals. What is important is the product of the signal and the weight: if that is positive, the signal is exhibitory (tends to drive the output positive); and if it is negative, the signal is inhibitory (tends to drive the output negative). All the input weight products are summed together to provide the internal activation of the neurone. The output from the neurone is some generally non-linear function of this internal activation.

By way of illustration, consider the almost trivial neurone shown in Figure Two. It has two user inputs and one output. The fixed bias is of no impotance, as its weight is zero. The output transfer function is simple: it consists of multiplying by one! Despite being almost trivial, such a neurone can distinguish between two types of input. One pair of input values are +0.75 and -0.25. For this input, the neurone is to output 0.5. There are many combinations of weights that will satisfy this requirement. However, it is also required

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."

that the neurone outputs 1.4 when presented with inputs of 0.5 and 0.1. Only one combination of weights satisfies both these requirements, as can be obtained from algebra in this simple case. Note that neither weight can be identified with one particular value of output, it is the interaction of weights and input values that produces the output. Figure Three shows the possible values of input for which the output is within, say, 20% of the correct value. Given a range of inputs (patterns), our neurone can sort them into two categories. Of course, if you give it some intermediate value of input (a pattern it has never seen before), you get a nonsense answer. The two categories shown in Figure Three are linearly separable; that is, they can be separated by a straight line drawn in space (and learned by a neurone with a linear transfer function).

By adding more inputs and a non-linear output transfer function (often either a sigmoid or tanh function), far more complex multi-dimensional decision surfaces can be generated. Of course, in this case we probably don't know the correct weight values before we start, so we have to train the neurone(s) by example. We initialize the weights to small random values around zero (not all identically zero!). Then we present a known example of pattern A at the inputs and calculate the output, which in all probability will be wrong. We adjust the weights individually to make it a little less wrong—if the signal weight product is driving the output away from the desired answer,

Figure One. A simulated neurone has input, weights, and output.

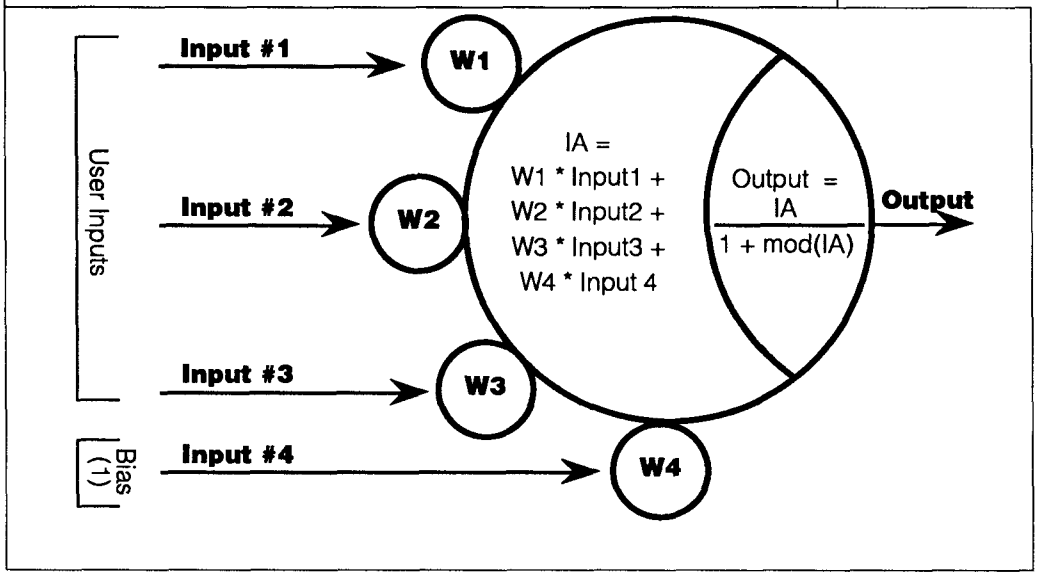
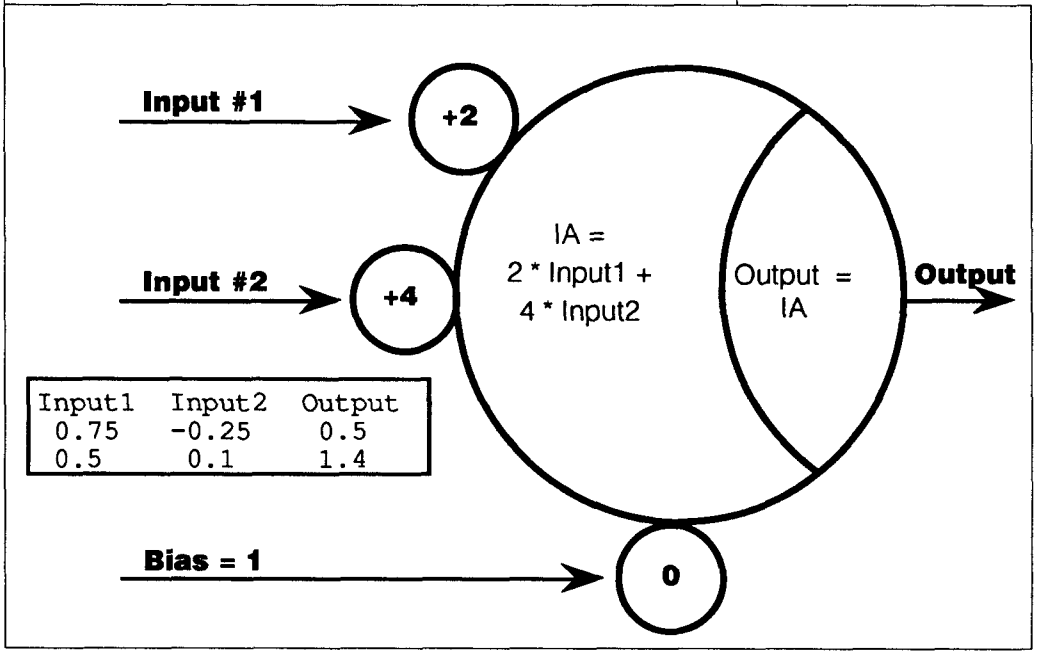
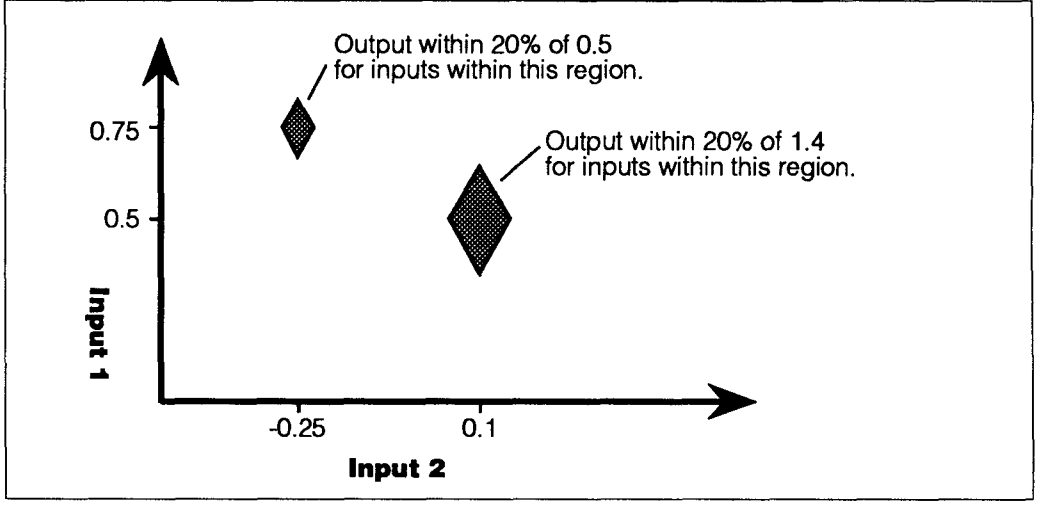


Figure Two. Weights and input values determine output.



Input1	Input2	Output
0.75	-0.25	0.5
0.5	0.1	1.4

Figure Three. Input that will result in the correct output (±20%).

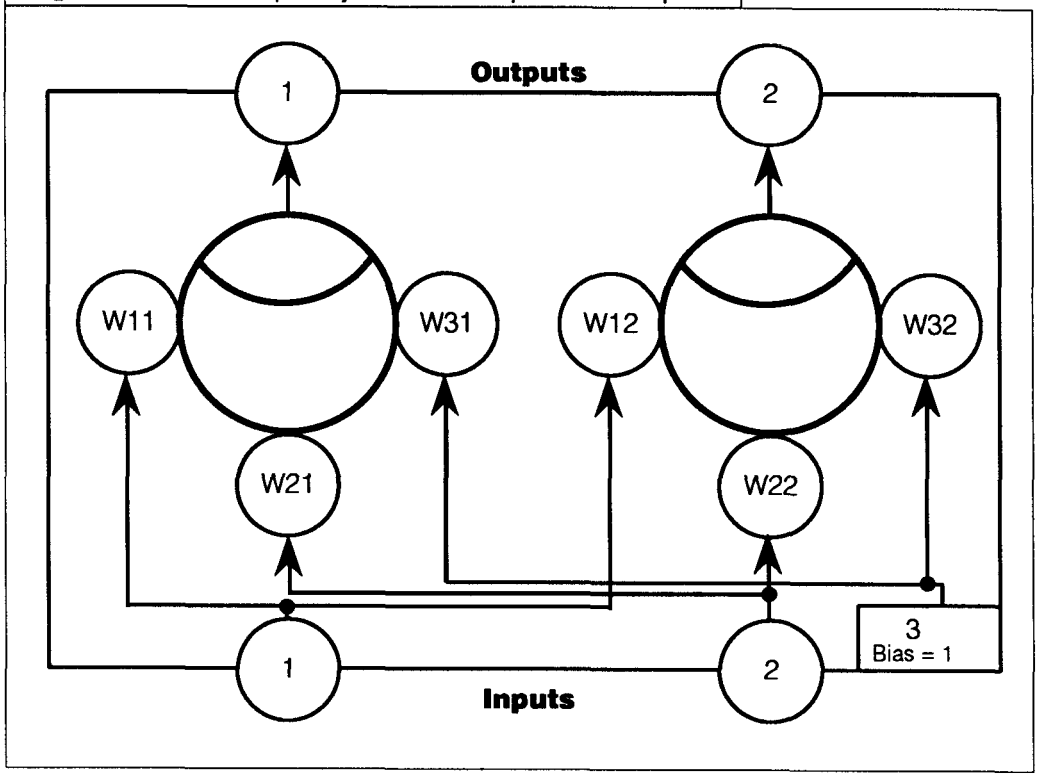


we decrease the weight slightly. If it is driving it towards the desired answer, we increase the weight slightly. Then we show it an example of pattern B and again adjust the weights.

This is repeated over and over again with many examples of all the patterns we wish to have learned. As time goes on, the outputs for each of the pattern types should become closer and closer to the ideal answer and the magnitude of the correction required to the weights should become smaller and smaller. This assumes that the neurone is capable of making the number of distinctions we ask of it and does not get stuck in any local-best solution on the way and be unable to get to a global-best solution. If this latter happens, adding small random values to all the weights may "jog" the neurone out of the local solution so that it is able to proceed to the desired global solution.

There is a limit to how much one neurone can do, and vastly enhanced performance is obtained by connecting numbers of neurones into networks. A network will often consist of neurones arranged in layers, with the output of one layer feeding forward to become the input to the next layer. The number of neurones may vary from layer to layer, and each input to a layer is connected to every neurone in the layer. Figure Four shows a simple layer with two inputs and two outputs. Note that it is not required that the number of inputs and the number of outputs be equal. Figure Four also shows the nomenclature used to identify the weights that will be used in the software. The bias input is connected to each neurone but is not considered an input to the layer.

Figure Four. A simple layer with two inputs and outputs.



The basic training method is still by example; the error at an output of a neurone is assumed to be caused equally by each input to that neurone, and the input weights to that neurone are altered to diminish the error at the output. The error at that input is back-propagated to the preceding layer. The error at the output of a neurone in the preceding layer is the sum of all the input errors at the inputs it feeds in the layer above. Once this is calculated, the weights on the inputs of the preceding layer can be altered.

The learning rate controls how fast the network learns: too small and the network will take forever to train, too large and the network will be bounced around from one position to another without ever having a chance to settle.

Formulae Used

The actual formulae used are given below; the derivation is beyond the scope of this paper.

First, some nomenclature. Let I_i be the i^{th} input to this layer, O_i be the i^{th} output from this layer, and let E_i be the error in the i^{th} output in this layer.

It is important to realise that the basic building block is the layer of neurones, not the single neurones themselves.

Forward Calculation

The internal activation of the n^{th} neurone in a layer due to the m inputs and the bias input (which is always 1) is

$$w_{n1}*I_1 + w_{n2}*I_2 + \ldots (w_{nm} * I_m)$$

where w_{n1} is the weight from the first input to the nth output, W_{n2} is the weight from the second input to the nth output, and so on. The actual output is the non-linear transform of this internal activation. In these examples, the transform used is

F(A) = A / (1+modA)

Backward (Training) Calculation

This involves two different calculations: the changes to the weights in the layer and the calculation of the errors to be back propagated to train the preceding layer. For simplicity of implementation, the back-propagated errors are calculated before the weights are updated.

The correction to the weight from the m^{th} input to the n^{th} output from the layer is

$$lcoeff * IOE_n * I_m$$

IOE_n is shorthand for $(1/(1+modO_n)^2) * E_n$

and

$(1/(1+modO_n)^2)$ is the derivative of the non-linear transform given above when the output is O_n.

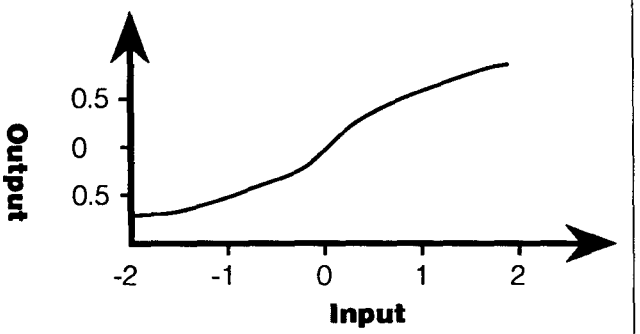
The error propagated back to the m^{th} input (and which will be the error of the m^{th} output of the previous layer) is

$IOE_1 * W_{m1} + IOE_2 * W_{m2} + \ldots + IOE_n * W_{mn}$

Implementation in Forth

When factorizing neural networks for implementation in Forth, it is important to realise that the basic building block is the layer of neurones, not the individual neurones themselves. Accordingly, a new defining word is produced to enable layers to be readily constructed. Information is moved to a layer, and the parameter stack receives output from a layer. The total number of major words that need to be defined is five. The most important ones are: one to define layers, one to initialize a specified layer, one to evaluate the internal activation at the outputs from a specified layer, one to calculate and store the errors at the outputs from a layer and, finally, one to perform one training weight update on a specified layer. These words are defined by:

Figure Five. Approaching the curve limit.




```
LAYER
Compile time:
  ( #outputs #inputs -- )
Run time:
  ( -- ladr )
```

Define a normal non-linear feed forward layer, allocating space for its weights, its current output, and its current input. At run time, return the address of this layer storage area (ladr).

```
INITIALIZE
( ladr -- )
```

Initialize a specified layer by setting all its weights to small random values around zero.

```
COMPUTE
( inn inn-1 ... in1 ladr -- )
```

Forward evaluate a normal non-linear feed forward layer by presenting the inputs in_1 through in_n to the network specified by the address adr and calculating the values of internal activation at out_1 through out_m from the layer.

```
CALC-LOAD-ERRORS
( Cn Cn-1 ... C1 ladr -- )
```

Given the "correct" answers on the stack ($C_1 - C_n$), this word calculates the errors at the outputs of the specified layer and stores them internally, ready for training

```
TRAIN
( learning-rate ladr -- ien
ien-1 ... ie1 )
```

Perform one training update of the weights in the layer specified by ladr, using the current error at each of the outputs previously stored. In the training update, use the values of the two learning coefficients lcoeff1 and lcoeff2 provided on the stack. Return the latest estimate of the error at the inputs (the ie_ns), which will form the estimated output error from the layer that fed this one during forward evaluation.

The Examples

The first example uses just a single layer to learn to distinguish between two type of input. An input pair both consisting of 0.5 should produce an output of -0.5. However, an input pair of -0.5 and 1 should produce an output of 0.5. The network learns this fast, as can be seen by running `Ex1`. A moment's thought will reveal that the second input is redundant in this case—just using a weight of -1 and the first input will give the correct answer. Checking how the network has solved the problem with `TEST .LAYER` will show that the network didn't see this and has used both inputs and the bias. A general rule: a network will normally use all the resources it has rather than

adopt a minimal solution. It is up to the user to estimate (by experimentation, if necessary) the best number of and size of the layers and how they should be interconnected.

The remaining examples are all multi-layer networks that learn the exclusive-or relationship. This is a very difficult problem for them to learn (and, of course, simpler methods exist for implementing XOR) but, owing to the low number of cases, it is a suitable example. Example Two uses two layers. The first has two inputs (and the bias) and three outputs. The second has three inputs (plus the bias) and one output. The output of the first directly feeds the second. Type Ex2 to see it learn; the answers it is trying to get are 0, 1, 0, and 1. It will take about 1000 training cycles before the outputs look at all like that. Even then, the 1s are really 0.8s. Looking at the transfer function we are using (Figure Five), it will take forever for the output to reach 1 as this is the limiting value of the transfer curve. To prevent this being a problem, input and output data are usually scaled to lie in the active region of the transfer function. Scaling the real-life input values to lie between plus and minus one, and the output values to lie between plus and minus 0.5, will result in better network performance. This will ensure that one stays away from the regions where the gradient is so small that no significant learning can occur. Example three is exactly the same as example two, except that the inputs and outputs have been scaled (0 becomes -0.5 and 1 becomes +0.5). Run Ex3 and note that all outputs are within 0.01 after only 300 cycles.

Finally, example four differs from example three only in that a different network geometry is used. Since, in this implementation, the layer interconnection is handled by the data placed on the parameter stack, any network geometry can be readily built. The network layers in this example are simpler, with the two inputs being presented to a layer with just one output. This output, along with the two network inputs, is then presented to a layer with three inputs and one output. A total of two nodes and seven weights only. Still, the network learns as well as example three (a network with four nodes and 13 weights), within 400 cycles.

This use of the minimum nodes and, therefore, weights is important. In real-life recognition problems with neural networks, the number of different training cases needed to train a network depends on how "noisy" the data is and how many weights (including those to the bias input) are to be trained. For exact data, such as the XOR, almost no cases per weight are needed. As the data gets noisier and you wish your network to extract and recognise the underlying pattern, this figure rises to at least 25 examples per weight.

Conclusion

The words presented here will allow any non-linear feed-forward network to be built and trained. They form the nucleus of a neural network simulation language, and demonstrate yet again the power of Forth for developing special languages for special tasks. They should allow artificial neural networks to be used as a tool on embedded processors as well as on resource-rich computers. Networks trained on one machine can be transferred, by noting the weights used, to any other machine.

These words allow artificial neural networks to be used as a tool on embedded processors as well as on resource-rich computers.

Figure Six. Required math functions (32MATH.SEQ).

```
Comment: This is a simple fixed point maths pack.  Each number is two words
(32 bits) with a 16 bit characteristic and a 16 bit mantissa.  This allows for
numbers up to approx + or - 32000 with a resolution of at least 0.0001.
Addition and subtraction are done with the normal double length words D+ and
D-, DABS operates correctly on these scaled numbers.  Multiplication or
division of two scaled numbers requires the words S* and S/.  S/ only gives an
answer accurate to about 5 digits - quite adequate for artificial neural
networks.  The double number on the top of the stack is converted into scaled
form by S#.  Note this only works for numbers just entered for which the
variable DPL is set up.  The scaled number on the top of the stack is printed
by S. always with four digits after the decimal point.
Three words are included especially for artificial neural networks, SRANDOM
returns a (fairly) random number in the range from 1 to -1, SFN(X) performs
the non-linear transform X / (1+mod(X)) and S'FN(X) calculates the derivative
of the transform function SFN(X) at the value X. SFN(X) is used instead of the
sigmoid or tanh functions as it and it's derivative are simple to compute.
comment; \

\ ************************** VARIABLE ************************************
VARIABLE RND here rnd !          \ space for current random number seed
\ ************************************************************************

\ Compute the sign of the answer to a scaled * or /
: GET-SIGN ( d1 d2 -- absd1 absd2 sign )
  dup 0< >r dabs                 \ sign of d2 to return stack, absd2 left
  2swap dup 0< >r dabs           \ do the same to d1
  2swap r> r> <>                 \ calculate sign of answer, true=negative
;
\ Unsigned 32bit * 32 bit multiply to give 32 bit answer. Checks for overflow.
\ Uses identity (an+b)*(cn+d)=acnn+bcn+adn+bd where n in this case is FFFFhex
: S* ( s1 s2 -- s3 )
  get-sign >r                    \ sign of answer to return stack
  rot dup >r over >r >r          \ stack parameters on the return stack
  over >r rot dup >r swap >r     \ .. in the order we will need them
  u*d 0 r> r> u*d d+             \ adn+bd
  r> r> u*d d+ 0 r> r> u*d d+    \ acnn+bcn+adn+bd
  0 <> abort" S* overflow"       \ We overflowed if top 16 bits not = 0
  rot drop r> if dnegate then    \ otherwise  update sign
;
: S/ ( s1 s2 -- s3 )
  2dup d0=                       \ Divisor S2 = zero?
  abort" S/ by 0!"               \ get out of here if so
  2over d0= if 2swap 2drop       \ answer is zero if s1=0
  else get-sign >r               \ work out sign of answer and save
    2swap -16 >r
    begin dup 16364 <
    while d2* r> 1+ >r           \ shift s1 as far left as possible
    repeat 2swap
    begin dup 0>
    while d2/ r> 1+ >r           \ reduce s2 to a 16 bit quantity
    repeat
    drop mu/mod rot drop         \ calculate the answer, except for powers of 2
    r> dup 0<>                   \ non-zero power of 2 to adjust by?
    if dup 0>
      if 0 do d2/ loop           \ do positive adjustment
      else abs 0 do d2* loop     \ do negative adjustment
      then
```

(Continued.)

HARVARD SOFTWORKS

NUMBER ONE IN FORTH INNOVATION

(513) 748-0390 P.O. Box 69, Springboro, OH 45066

MEET THAT DEADLINE ! ! !

- Use subroutine libraries written for other languages! More efficiently!
- Combine raw power of extensible languages with convenience of carefully implemented functions!
- Yes, it is faster than optimized C!
- Compile 40,000 lines per minute!
- Stay totally interactive, even while compiling!
- Program at any level of abstraction from machine code thru application specific language with equal ease and efficiency!
- Alter routines without recompiling!
- Use source code for 2500 functions!
- Use data structures, control structures, and interface protocols from any other language!
- Implement borrowed feature, often more efficiently than in the source!
- Use an architecture that supports small programs or full megabyte ones with a single version!
- Forget chaotic syntax requirements!
- Outperform good programmers stuck using conventional languages! (But only until they also switch.)

HS/FORTH with FOOPS - The only full multiple inheritance interactive object oriented language under MSDOS!

Seeing is believing, OOL's really are incredible at simplifying important parts of any significant program. So naturally the theoreticians drive the idea into the ground trying to bend all tasks to their noble mold. Add on OOL's provide a better solution, but only Forth allows the add on to blend in as an integral part of the language and only HS/FORTH provides true multiple inheritance & membership.

Lets define classes BODY, ARM, and ROBOT, with methods MOVE and RAISE. The ROBOT class inherits:

```
INHERIT> BODY
HAS> ARM RightArm
HAS> ARM LeftArm
```

If Simon, Alvin, and Theodore are robots we could control them with:

```
Alvin 's RightArm RAISE          or:
+5 -10 Simon MOVE                or:
+5 +20 FOR-ALL ROBOT MOVE
```

Now that is a null learning curve!

WAKE UP ! ! !

Forth is no longer a language that tempts programmers with "great expectations", then frustrates them with the need to reinvent simple tools expected in any commercial language.

HS/FORTH Meets Your Needs!

Don't judge Forth by public domain products or ones from vendors primarily interested in consulting - they profit from not providing needed tools! Public domain versions are cheap - if your time is worthless. Useful in learning Forth's basics, they fail to show its true potential. Not to mention being s-l-o-w.

We don't shortchange you with promises. We provide implemented functions to help you complete your application quickly. And we ask you not to shortchange us by trying to save a few bucks using inadequate public domain or pirate versions. We worked hard coming up with the ideas that you now see sprouting up in other Forths. We won't throw in the towel, but the drain on resources delays the introduction of even better tools. Don't kid yourself, you are not just another drop in the bucket, your personal decision really does matter. In return, we'll provide you with the best tools money can buy.

The only limit with Forth is your own imagination!

You can't add extensibility to fossilized compilers. You are at the mercy of that language's vendor. You can easily add features from other languages to **HS/FORTH**. And using our automatic optimizer or learning a very little bit of assembly language makes your addition zip along as well as in the parent language.

Speaking of assembly language, learning it in a supportive Forth environment turns the learning curve into a light speed escalator. People who failed previous attempts to use assembly language, conquer it in a few hours or days using HS/FORTH.

HS/FORTH runs under MSDOS or PCDOS, or from ROM. Each level includes **all** features of lower ones. Level upgrades: $25. plus price difference between levels. Source code is in ordinary ASCII text files.

All HS/FORTH systems support full megabyte or larger programs & data, and run faster than any 64k limited ones even without automatic optimization -- which accepts almost anything and accelerates to near assembly language speed. Optimizer, assembler, and tools can load transiently. Resize segments, redefine words, eliminate headers without recompiling. Compile 79 and 83 Standard plus F83 programs.

PERSONAL LEVEL $299.
NEW! Fast direct to video memory text & scaled/clipped/windowed graphics in bit blit windows, mono, cga, ega, vga, all ellipsoids, splines, bezier curves, arcs, turtles; lightning fast pattern drawing even with irregular boundaries; powerful parsing, formatting, file and device I/O; DOS shells; interrupt handlers; call high level Forth from interrupts; single step trace, decompiler; music; compile 40,000 lines per minute, stacks; file search paths; format to strings. software floating point, trig, transcendental, 18 digit integer & scaled integer math; vars: A B * IS C compiles to 4 words, 1..4 dimension var arrays; **automatic optimizer for machine code speed.**

PROFESSIONAL LEVEL $399.
hardware floating point - data structures for all data types from simple thru complex 4D var arrays - operations complete thru complex hyperbolics; turnkey, seal; interactive dynamic linker for foreign subroutine libraries; round robin & interrupt driven multitaskers; dynamic string manager; file blocks, sector mapped blocks; x86&7 assemblers.

PRODUCTION LEVEL $499.
Metacompiler: DOS/ROM/direct/indirect; threaded systems start at 200 bytes, Forth cores from 2 kbytes; C data structures & struct+ compiler; TurboWindow-C MetaGraphics library, 200 graphic/window functions, PostScript style line attributes & fonts, viewports.

ONLINE GLOSSARY $ 45.

PROFESSIONAL and PRODUCTION LEVEL EXTENSIONS:

FOOPS+ with multiple inheritance **$ 79.**
TOOLS & TOYS DISK $ 79.
286FORTH or **386FORTH $299.**
16 Megabyte physical address space or gigabyte virtual for programs and data; DOS & BIOS fully and freely available; 32 bit address/operand range with 386.
ROMULUS HS/FORTH from ROM **$ 99.**
FFORTRAN translator/mathpak **$ 79.**
Compile Fortran subroutines! Formulas, logic, do loops, arrays; matrix math, FFT, linear equations, random numbers.

Shipping/system: US: $7. Canada: $19. foreign: $49. We accept MC, VISA, & AmEx

```
    else drop
    then
    r> if dnegate then          \ finally update the sign of the answer
  then
;
: S. ( s# -- )                  \ print a scaled number
  dup 0<                        \ negative?
  if dabs ascii -               \ yes load a minus and get absolute value
  else ascii +                  \ no, just laod a +
  then emit                     \ show sign
  (.) type ." ."                \ print integer part
  10000 u*d nip                 \ get mantissa
  (u.) 4 over - dup 0>          \ convert to string, < four digits in string?
  if 0 do ascii 0 emit loop     \ if so output correct number of leading zeros
  else drop
  then type bl emit             \ end with a blank to keep it pretty
;

\ Convert number just put on stack to scaled form
: S# ( # -- s# )
  dup 0< >r                     \ save sign
  dpl @ -1 =                    \ no decimal places entered?
  if abs 0 swap                 \ just make it double integer if so
  else dabs dup 0<>             \ too many digits entered?
    if abort" S# too long" then \ if so tell them and abort
    swap dpl @
    case                        \ shift left 16 bits and scale it
      1 of 10 mu/mod rot drop endof     \ 1 divide by 10
      2 of 100 mu/mod rot drop endof    \ 2 divide by 100
      3 of 1000 mu/mod rot drop endof   \ 3 divide by 1000
      4 of 10000 mu/mod rot drop endof  \ 4 divide by 10000
      Abort" Eoverflow"                 \ anything else = overflow
    endcase
  then r> if dnegate then       \ ensure we have the correct sign
;

: SRANDOM ( -- Srand# )
  rnd @                         \ get last random number
  31421 * 6927 +                \ generate a new not very random 16 bit number!
  dup rnd !                     \ update record of last random number
  s>d                           \ scale it into the range from 1 to -1 approx
;

: SFN(X) ( X -- fnX )           \ calculate the non-linear function (X/1+modX)
  0 1 s* 2dup dabs 0 1 d+ s/
;

: S'FN(X) ( X -- s'fnX )        \ calculate the derivative of SFN(X)
  dabs 0 1 d+ 2dup s*           \ (1+modX)(1+modX)
  0 1 2swap s/                  \ calc (1 / (1+modX)(1+modX))
;
```

(Continued.)

```
\ ************************ SOME USEFUL VALUES ******************************
  0   s# 2CONSTANT S0
  0.3 s# 2CONSTANT S0.3          -0.3 s# 2CONSTANT S-0.3
  0.5 s# 2CONSTANT S0.5          -0.5 s# 2CONSTANT S-0.5
  0.8 s# 2CONSTANT S0.8          -0.8 s# 2CONSTANT S-0.8
  1.0 s# 2CONSTANT S1            -1.0 s# 2CONSTANT S-1
\ ****************************************************************************
```

Figure Seven. Basic neural net construction kit (BASICNN.SEQ).

```
Comment: This file contains words that allow the construction, initialization,
use and inspection of non-linear feedforward artificial neural networks
together with their training using the back propagation algorithm.
We need the following file to have been loaded, check and load if not.
comment;
\ ****************************************************************************
                ( ====> )     NEEDS 32MATH.SEQ      ( <==== )
\ ****************************************************************************
comment:         Structure built by a neural layer word
Starting Offset (words)   Item                          Length (words)
0                         m (#inputs)=user#+1           1
1                         n (#outputs)                  1
2                         m weights to node 1           2m
2m+2                      m weights to node 2           2m
...
2(n-1*m)+2                m weights to node n           2m
2(n*m)+2                  current outputs               2n
2(n*m)+2+2n               current inputs                2m (bias last,fixed at 1)
2(n*m)+2+2n+2m            output errors (int form)      2n
2(n*m)+2+4n+2m = (m+2)(2n+1)+m
A bias input is automatically added to each layer specified by the user.
To the user inputs & outputs are counted from 1,internally they start at 0
comment;
\ ****************************************************************************
comment:              Main words defined in this file
 LAYER  ( #out #in -- )          a layer defining word
 INITIALIZE ( ladr -- )          randomises all layer weights, set bias = 1,
                                 all else = 0
 .LAYER ( ladr -- )              prints a snapshot of inside a layer
 COMPUTE ( INm INm-1 ... IN1 ladr --  )
                                 calculates the internal activations at outputs
                                 from specified layer and stores internally
 GET-OUTPUT ( n -- S# )          gets the nth actual output from current layer
 GET-OUTPUTS ( ladr -- OUTn OUTn-1 ... OUT1 )
                                 places actual outputs from this layer on the
                                 stack ready for processing by the next layer.
 LOAD-ERRORS ( OEn OEn-1 ... OE1 ladr -- )
                                 loads output errors ready for training
 CALC-LOAD-ERRORS ( Cn Cn-1 ... C1 ladr -- )
                                 calculates and stores the output errors using
                                 the 'correct' values Cx from the stack
 TRAIN  ( lrate ladr -- IEm IEm-1 .... IE1 )
                                 trains a layer,leaves back propagated errors
                                 on stack ready to train the previous layer
comment;
```

(Continued.)

```
\ ******************************************************************************
\ ************************* VARIABLES ******************************************

VARIABLE 'LADR                  \ start address of the layer in use
VARIABLE CLIPPING               \ output errors to be clipped?
VARIABLE ICOUNT                 \ will keep track of itteration number

\ ******************************************************************************
: LAYER                         \ A layer defining word

  create                        \ compile operation stack ( #out #in -- )
  1+ 2dup                       \ allow for bias input n m and keep copies
  here >r                       \ save start address of array
  dup 2+ rot dup +  1+ * +      \ space needed in words
  2 * allot                     \ make room in bytes
  r> tuck !                     \ save #inputs
  2+ !                          \ save #outputs

  does>                         \ run time operation stack ( -- ladr )
;

: LADR 'ladr @ ;                \ return start address of layer
: #INPUTS ladr @ ;              \ number of inputs in this layer
: #OUTPUTS ladr 2+ @ ;          \ number of outputs from this layer

\ get adr of weight Wmn joining mth input to nth output in current layer
: GET-WADR ( out# in# -- weight-adr )
  1- swap 1- #inputs * + 2* 2+          \ offset in words
  2* ladr +                             \ actual address
;

\ get adr of latest value of internal activation of nth output of current layer
: GET-INTACT ( out# -- output-adr )
  1- 2* #outputs #inputs * 2* 2+ +      \ offset in words
  2* ladr +                             \ actual address
;

\ get adr of latest value of the mth input to a layer
: GET-INADR ( in# -- input-adr )
  get-intact #outputs 4 * +             \ actual address
;

\ get adr of internal form of latest output error at nth output of current layer
: GET-OEADR ( out# -- outerror-adr )
  get-inadr #inputs 4 * +               \ actual address
;

: INITIALIZE ( ladr -- ) \ randomise all weights, set bias = 1, all else = 0
  'ladr !                       \ save layer start address
  #outputs 1+ 1 do
  #inputs 1+ 1  do srandom      \ set up loops and
  j i get-wadr 2! loop loop     \ randomise all weights
  #outputs 1+ 1                 \ adr-first-output #outputs 0
  do 0. i get-intact 2! loop    \ zero all outputs
  #inputs  1                    \ adr-first-input #inputs-1 0
  do 0. i get-inadr 2! loop     \ zero all but bias input
```

(Continued.)

```
  0 1 #inputs get-inadr 2!       \ set bias=1
  #outputs 1+ 1                  \ adr of first output-error #output-errors 0
  do 0. i get-oeadr 2! loop      \ zero all output errors
;

\ print a snapshot of inside a layer
: .LAYER ( ladr -- )
  'ladr ! crlf                                    \ save layer start address
  ladr 3 - >name.id ." a neural layer of "        \ print layer name
  #inputs 1- . ." inputs and "                    \   number of inputs
  #outputs . ." output(s)" crlf                   \   and number of outputs.
  ." Outputs " #outputs 1+ 1
  do i get-intact 2@ sfn(x) s. loop crlf          \ print outputs
  ." Int-act " #outputs 1+ 1
  do i get-intact 2@ s. loop crlf                 \ print internal activation
  ." IntOerr " #outputs 1+ 1
  do i get-oeadr 2@ s. loop crlf                  \ print internal output errors
  ." Weights between inputs and...." crlf
  #outputs 1+ 1
  do ." Output" i .
    #inputs 1+ 1
    do j i get-wadr 2@ s. loop crlf               \ print weights
  loop
  ." Inputs  " #inputs 1+ 1
  do i get-inadr 2@ s. loop ." (Bias)" crlf       \ print inputs
;

\ calculate internal activation at outputs from specified layer and store
: COMPUTE ( SINm SINm-1 ... SIN1 ladr --  )
  'ladr ! #inputs 1                       \ save layer adr, get # inputs
  do i get-inadr 2! loop                  \ save each input value in turn
                                          \ calculate internal activations
  #outputs 1+ 1 do                        \ loop over each output in turn
    0.  #inputs 1+ 1 do                   \ and each input in turn
      i get-inadr 2@ j i get-wadr 2@      \ get input and weight
      s* d+ loop                          \ update internal activation and loop
    i get-intact 2!                       \ save int activation from this node
  loop
;

\ Place nth output on the stack. If clipping is true, clip so that
\ outputs >0.5 are set to 0.5, outputs <-0.5 are set to -0.5
: GET-OUTPUT ( n -- S# )
  get-intact 2@ sfn(x)                    \ place actual output on stack
  clipping @                              \ clipping wanted?
  if s0.5 dmin s-0.5 dmax then            \ if so, clip if > 0.5 or < -0.5
;

\ Place all this layers outputs on stack ready for processing by next layer.
: GET-OUTPUTS ( adr -- OUTn OUTn-1 ... OUT1 )
  'ladr ! 1 #outputs                      \ need to get last output first!
  do i get-output -1 +loop                \ loop until all on stack
;
```

(Continued.)

```
: LOAD-ERRORS ( OEn OEn-1 ... OE1 ladr -- ) \ Load errors ready for training
  'ladr ! #outputs 1+ 1              \ save layer address, work out # to do
  do i get-oeadr 2! loop             \ store errors in correct places
;

\ Given the correct outputs on the stack, calculate the error in the current
\ outputs and store internally as the output errors.
: CALC-LOAD-ERRORS ( Cn Cn-1 ... C1 ladr -- )
  'ladr ! #outputs 1+ 1              \ save layer address, work out # to do
  do                                 \ each output in turn
    i get-output d- i get-oeadr 2!   \ calculate and store error
  loop
;

\ Train a layer.  Latest estimate of the output errors from this layer must
\ have been stored internally before entry.  Back-propagated errors at inputs
\ to this layer are on the stack on exit.  Layer weights are updated.
: TRAIN  ( lrate ladr -- IEm IEm-1 .... IE1 )
  'ladr ! >r >r                      \ save layer address & learning rate
  #outputs 1+ 1 do                   \ do each output in turn
    i get-intact 2@ s'fn(x)          \ gradient at output
    i get-oeadr 2@ s* i get-oeadr 2! \ convert output error to internal form
  loop
  1 #inputs 1- do                    \ loop to calc back propagated errors
    0. #outputs 1+ 1                 \ loop over each contributing weight
    do i get-oeadr 2@                \ get ith error
      i j get-wadr 2@                \ & weight joining ith output & jth input
    s* d+ loop                       \ update sum & go get next contribution
  -1 +loop                           \ leave answer on stack,go do next input
  r> r> #inputs 1+ 1                 \ retrieve learning rate & calc # inputs
  do #outputs 1+ 1                   \ cal # outputs
    do i get-oeadr 2@ j get-inadr    \ get output-error & input address
      2@ s* 2over s* i j             \ calculate delta-weight
      get-wadr dup >r 2@ d+ r> 2!    \ calculate and save new-weight
    loop                             \ loop to do next weight
  loop 2drop                         \ loop to do next output or lose lrate
;
```

Figure Eight. Saving and loading neural network layers (NNDEMO.SEQ).

```
Comment:
This is an add on file which provides a word (NL->DISK) to save the contents
of a neural layer to disk and a word to retrieve these contents from disk
(DISK->NL).  A layer with the correct number of inputs and outputs must have
been created before attempting to load the layer, no check is made to see if
this has been done.
The syntax is:-
        <layer-name> SAVE-LAYER <file-name>              and
        <layer-name> LOAD-LAYER <file-name>
It also contains words to handle network inputs from disk.  These words are
SET-SCALES which works out the scale factors and sets up the scaling variables
(these can be saved to or read from disk with SAVE-SCALES and READ-SCALES) and
TEACH-NETWORK which is a generic network training word that expects the
training file on disk and trains for the number of cases specified on the
```

(Continued.)

```
stack on entry, or, if no number is specified, for 1000 cases.  It makes use
of four deferred words.  One, FORWARD-WORD, must take the network inputs from
the stack and work out the network output.  A second TRAIN-WORD will take the
correct answers from the stack and make one training pass backwards through
the network.  SHOW-INFO will display the state of the network and DO-INIT will
perform any initialization required before training is commenced. Appropriate
words for the network you are using must be assigned to these deferred words.
The syntax is:-
          SET-SCALES    <file-name>
          SAVE-SCALES   <file-name>
          LOAD-SCALES   <file-name>
        n TEACH-NETWORK <file-name>
The file BASICNN.SEQ must be loaded before this file.
comment;
\ ******************************************************************************
                ( ===> ) NEEDS BASICNN.SEQ ( <=== )
\ ******************************************************************************
\ ********************** VARIABLES AND DEFERRED WORDS ************************
        CREATE I-O 16 ALLOT \ space for four double variables .....
           \ input offset  (I-O) 4 bytes  |--------------------------------|
           \ input scale   (I-S) 4 bytes  | Used to scale inputs and output |
           \ output offset (O-O) 4 bytes  |   to and from internal values   |
           \ output scale  (O-S) 4 bytes  |--------------------------------|
        : I-S i-o 4 + ;   : O-O i-o 8 + ;   : O-S i-o 12 + ;
        VARIABLE N-IN          \ number of inputs to network
        VARIABLE N-OUT         \ number of outputs from network
        VARIABLE MAX#          \ maximum number of training cases to do
        DEFER FORWARD-WORD     \ word to do one forward pass of this network
        DEFER TRAIN-WORD       \ word to do one training pass of this network
        DEFER DISPLAY-INFO     \ word to display outputs or errors or ...
        DEFER DO-INIT          \ word to do any extra initialization required
        DEFER (PROCESS-FILE) \ will point to the actual processing to be done
\ ******************************************************************************
: OPEN-FILE
  seqhandle+ hopen              \ open file with name in current handle + 1
  0<> abort" Can't open it!"    \ abort with message if error
;

: MAKE-FILE ( -- )              \ make file with name in current handle + 1
  seqhandle+ hopen              \ try to open the file
  0 = if                        \ does it already exist?
    crlf ." O.K. to overwrite old " \ yes, ask if we can overwrite...
    seqhandle+ count type ." ?" \ show name
    begin key? until key        \ get reply
    UPC ascii Y <>              \ permission not granted?
    if seqhandle+ hclose        \ close file if so and
      abort" Untouched!"        \ put their mind at rest
    then
  then
  seqhandle+ hcreate 0 <>       \ create the file, any problem?
  0<> abort" Can't create it!"  \ abort with message if so
;

: CALCULATE-SIZE  ( ladr -- n )
  dup 2+ @ swap @               \ get # outputs (n) and inputs (m)in layer
  dup 2+ rot dup + 1+ * + 2*    \ calculate number of bytes in layer
;
```

(Continued.)

```
: LOAD-LAYER ( ladr -- )          \ read the specified file into a layer.
  seqhandle+ !hcb open-file      \ open new handle, get file name in open it
  dup                            \ keep copy of first address to load to
  calculate-size                 \ work out how many to load
  tuck                           \ save copy of how many we expect
  seqhandle+ hread               \ read the file, returns number of bytes read
  <> abort" Bad layer read! "    \ complain if it is not what we were expecting
  seqhandle+ hclose drop         \ else just close the file we were using
;

: SAVE-LAYER ( ladr -- )         \ write the specified layer into a file
  seqhandle+ !hcb                \ open new handle and put file name in it
  make-file                      \ make the file to write to
  dup                            \ keep copy of first address to load from
  calculate-size                 \ how many to save
  tuck                           \ save copy of how many we plan to save
  seqhandle+ hwrite              \ write file, return number of bytes written
  <> abort" Bad layer write!"    \ complain if it is not what we expected
  seqhandle+ hclose drop         \ else just close the file we were using
;

: PROCESS-FILE ( -- )
  bl word                        \ get file name to use
  filepointer 2@ outbuf c@ 0 d- \ calc where we got to in current file
  2>r                            \ save for now
  sequp                          \ move up one handle
  $file ?open.error              \ open file,reset input buffer,check if error
  0.0 seek                       \ point to start of file
        %save>  'tib             \ save address of tib we are using
  false %save!>  >in             \ save # already done in tib and set to 0
  false %save!> #tib             \ save total # to do and set to 0
  fillbuff                       \ load up with the first of the file
  (process-file)                 \ now do the actual processing
  %restore> #tib                 \ put things ..
  %restore> >in                  \     .. back as ..
  %restore> 'tib                 \       .. they were..
  2r> filepointer 2!             \ restore pointer into previous file
  seqdown                        \ close this file and go back to previous one
;

: MAX! ( s# adr -- ) dup>r 2@ dmax r> 2! ; \ save larger of s# or current max

: MIN! ( s# adr -- ) dup>r 2@ dmin r> 2! ; \ save smaller of s# or current min

: READ#                          \ read one scaled number from the TIB
  bl word number                 \ isolate one word & convert to double number
  double? not if drop then       \ make it a single if no decimal point entered
  s#                             \ convert to scaled number
;

comment:
Get maxia and minima for raw (direct off disk) input and output values, store
these for the time being, maxima in the appropriate offset and minima in
approprite scale.  Then calculate and load the offsets and scales variables.
Scale is internal range/(filemax-filemin), offset is (filemax+filemin)/2.
The internal range for inputs is from -1 to +1, for outputs from -0.5 to +0.5
```

(Continued.)

```
To scale our raw network input and output from/to the correct internal range
use   scaled input  = (raw input - in-offset ) * in-scale
and   scaled output = (raw output - out-offset ) * out-scale
comment;

: (SET-SCALES)          ( --- )
  i-o 16 erase                    \ initialize all max and min to zero
  0 begin                         \ keep count of number of values done
    filltib tib 1- c@  2 >        \ read line, length > 2 (crlf->blanks)?
  while
    n-in @ n-out @ + 0            \ set up to do each value on a line in turn
    do dup n-in @ n-out @ + mod   \ calculate place on line
      n-in @ >= >r                \ is value an output?
      1+ read# 2dup r>            \ update count,get two copies of next value
      if o-o max! o-s min!        \ update output maximum and minimum or
      else i-o max! i-s min!      \ ..update input maximum and minimum
      then
    loop
  repeat drop                     \ lose unwanted information
;

: SET-SCALES
  ['] (set-scales) is (process-file)
  process-file                    \ get the file maxima and minima
  0 2 i-o 2@ i-s 2@ d- s/         \ calculate input scale
  i-o 2@ i-s 2@ d+ 0 2 s/         \ calculate input offset
  i-o 2! i-s 2!                   \ store them
  0 1 o-o 2@ o-s 2@ d- s/         \ calculate output scale
  o-o 2@ o-s 2@ d+ 0 2 s/         \ calculate output offset
  o-o 2! o-s 2!                   \ store them
;
: LOAD-SCALES ( -- )              \ read the specified scaling factors from disk
  seqhandle+ !hcb                 \ open new handle and put file name in it
  open-file
  i-o 16 seqhandle+ hread         \ read the file, returns number of bytes read
  16 <> abort" Bad scale factors read! " \ complain if not what we wanted
  seqhandle+ hclose drop          \ close the file we were using
;
: SAVE-SCALES ( I-Oadr -- )       \ write the specified scaling factors to disk
  seqhandle+ !hcb                 \ open new handle and put file name in it
  make-file                       \ make the file to write to
  i-o 16 seqhandle+ hwrite        \ write file, return number of bytes written
  16 <> abort" Bad scale factors write!"  \ complain if it is not what we expected
  seqhandle+ hclose drop          \ close the file we were using
;

: SCALE-IN#     ( S# -- S# ) i-o 2@ d- i-s 2@ s* ;
: SCALE-OUT#    ( S# -- S# ) o-o 2@ d- o-s 2@ s* ;
: UNSCALE-OUT# ( S# -- S# ) o-s 2@ s/ o-o 2@ d+ ;

: (TEACH-NETWORK)
  begin 0.0 seek                          \ start at beginning of file
    begin
      filltib tib 1- c@  2 >              \ read line, is it > just crlf?
```

(Continued.)

```
    while                                   \ provided length is > 2
      n-in @ 0 do read# scale-in# loop      \ get a set of inputs
      forward-word                          \ calculate the outputs
      n-out @ 0 do read# scale-out# loop    \ get a set of outputs
      train-word                            \ calculate errors and train network
      display-info                          \ display how well we did
      -1 max# +! max# @ 0= key? or         \ done all or do they want out?
      if exit then                          \ leave if so
    repeat
  again
;

\ Train for n passes through the data set (or until a key is pressed)
: TEACH-NETWORK ( n -- )
  depth 0= if 1000 then max# !              \ 1000 cases unless told otherwise
  ['] (teach-network) is (process-file)
  do-init                                   \ do any extra initialization needed
  process-file                              \ now open the file and learn from it
;
```


Advertisers Index

FORML 8, 40

Forth Institute 26, 27

Forth Interest Group 30

Harvard Softworks 15

Laboratory Microsystems 13

Miller Microcomputer Services 12

Next Generation Systems 34

Saelig Company 24

Silicon Composers 2

An introduction to Pygmy Forth

Frank Sergeant
San Marcos, Texas

"Push your left foot down," is probably the answer a student driver wants when asking about a clutch; an engineer might need more detail. So, what do you want to know about Pygmy? How it works inside or how to put it in gear? I'll give an overview, some general tips, and explain how to use the editor and work with files.

Perhaps I'm mixing metaphors by talking about a clutch. Is Pygmy Forth more like an automobile or a person? The person metaphor suggests Pygmy is alive, has volition, can act as your agent. There is something to that, but I don't want a Forth to take too much initiative. I believe we should drive the computer language rather than letting it drive us, and this has a lot to do with "why Pygmy?" I picked the name to suggest "small but tough." I wanted a Forth done my way, and that was under my control.

Pygmy Forth version 1.3 is a fast, direct-threaded Forth for MS-/PC-DOS machines, complete with editor, assembler, and metacompiler. With editor and assembler, it can fit in a 14K byte file. The kernel alone fits in less than 8K. The entire system, including program, full source code, and documentation, is small enough to fit on a single 360K floppy. Pygmy is intended to be a complete Forth for real work. It can metacompile itself, simply, so you can change anything you don't like.

It comes with three main files other than the program itself. PYGMY.SCR is a block file containing all the source code. PYGMY.TXT is a text file containing the documentation. YOURFILE.SCR is an empty block file ready for your own code.

Frank Sergeant, the author and implementor of Pygmy Forth, is a hardware and software consultant specializing in business and real-time systems.

It is small enough to control and understand, yet powerful enough for real work.

Are We There Yet?

Pygmy is one step on my path toward a "perfect" Forth. It was inspired by cmFORTH for the NOVIX, written by Charles Moore. Much of the overall structure, and some of the specific high-level code, reflect this influence.

Among my goals were a faster, more comfortable editor; reduced size and complexity; and inclusion of certain cmFORTH ideas such as `PUSH POP` (instead of `>R R>`), `FOR ... NEXT`, and simple metacompilation.

Pygmy features include:

1. A fast screen-oriented block editor.
2. An 8088/8086 assembler.
3. Full source code.
4. Full metacompiler.
5. Up to 15 files open and accessible at one time.
6. Default set of files opened automatically.
7. `FOR ... NEXT`, `PUSH`, `POP`, `\`, `COMPILER` vocabulary, and other cmFORTH improvements.
8. Direct screen writes for speed on both monochrome and color systems.
9. Vectored I/O (`EMIT`, `KEY`, `KEY?`, `CR`).
10. *Starting Forth* compatibility hints for people new to Forth.
11. Source code for fast hashed dictionary searching.
12. "Compiles over 30,000 lines per minute."

How Fast Is It?

Pygmy is fast in several different ways. First, it is customized to run on MS-

DOS machines, with many words written in 8086 assembly language. Second, the editor is fast. This is important because much of your time developing applications is spent working with the editor. Third, a dictionary hashing system is available to do very rapid dictionary searches. This makes compiles fast.

The hashing is interesting because it is an add-on. The regular dictionary links are not disturbed, and the hashing can be turned on and off at will. This allows you to turn on the hashing while you are developing, but leave it out of the finished application. To find a word takes an average of less than two compares.

Speed Comparisons

One commercial Forth claims a compilation speed of 40,000 lines per minute. This sounds very impressive, but the ad doesn't say which processor or what clock rate. There could be a substantial difference in speed between an original 4.77 MHz PC and a 33 MHz '386. Furthermore, did the line count include blank lines? Did each line contain many Forth words (horizontal style) or only a single word (vertical style)? Anyway, it made me curious to see how Pygmy compared.

I used a stop watch to time how long my current experimental version of Pygmy took to metacompile its own kernel, with hashing turned on. I ran it out of a RAM disk, to eliminate the disk access time. I did a quick count of all the non-blank lines in the blocks that were actually loaded, including comments and any lines that were not compiled because of an

EXIT above them. The coding style is horizontal, with a number of Forth words on most lines. I counted 872 lines. The results are shown in Figure One.

I feel sure that with exact timing, plus editing the source code for speed, or by using a '486 machine, I could "cook" the figures to get a matching 40,000 lines per minute.

cmFORTH-isms

There are three name changes in cmFORTH (and Pygmy) that delight me:

new	*old*
PUSH	>R
POP	R>
\	[COMPILE]

For me, >R and R> require extra thinking each time to tell which is which, while PUSH and POP are immediately obvious.

Pygmy does a few things differently from cmFORTH. For example, FOR ... NEXT performs the body of the loop zero times for a count of zero, three times for a count of three, etc. In cmFORTH, a count of three would perform the body of the loop four times. I have been so happy using FOR ... NEXT that I do not include DO ... LOOP in the kernel. If you want it, though, you can load it—courtesy of Robert Berkey, from screen 171 of PYGMY.SCR.

Miscellaneous Tips

Set Caps Lock on. Most built-in words must be typed in upper case.

To abandon changes you have just made in the editor, use Esc to get out of the editor, then type EMPTY-BUFFERS.

Figure One. Speed test of experimental version.

	machine time in seconds	lines per minute
8 MHz 8088	14.5	3,608
10 MHz 8088	10.7	4,890
33 MHz 80386	1.5	34,880

?SCROLL is embedded in WORDS and DU to let you start and stop the display by pressing any key (except Esc). To bail out, whether you are scrolling or paused, press Esc. You can also put ?SCROLL into your own words. For my tastes, this is much better than the common practice of aborting when you press Enter.

DUMP (a -- a') and DU (a # -- a') allow you to inspect memory. DUMP dumps one line and leaves the address of the next line ready for typing DUMP once more. DU repeats DUMP for a number of lines. ?SCROLL is built in, so feel free to type 0 2000 DU (you can get out of it with Esc, or pause with any key). You'll probably want to say HEX first.

.FILES (--) shows the files that are currently open and the block numbers associated with them. You can open *any* type of file; you are not limited to Forth-style block files.

The word ." works either inside or outside of colon definitions. There is no need for the abomination . ((actually, there are two words named ." with one in FORTH and the other in COMPILER.)

Pygmy recognizes $xxxx as a hex number (e.g., $2000 or $1FFE), and it recognizes character literals as well (e.g., 'A, 'B, 'C, and 'z). The hex literals are a great convenience and allow us to stay in DECIMAL more of the time. The character literals avoid the need for the ugly CHAR and [CHAR] or ASCII and [ASCII].

NOT in Pygmy inverts the truth value on the stack. It is equivalent to 0=. If you want to invert each bit individually, use -1 XOR.

If you want an application to execute your code automatically (rather than coming up in Forth), just point the word BOOT at your application's highest-level word. Suppose you have named it YOUR-APPLICATION. Type:

```
' YOUR-APPLICATION IS BOOT
```

Then save it to disk with something like:

```
SAVE YOUR.COM
```

How Pygmy Handles Files

As my dissatisfaction with F83's editor and file handling with OPEN ... FROM was part of why I developed Pygmy, I've tried to figure out what specifically is wrong with the F83 file interface. The minor problem is its use of CP/M-style FCBs (file control blocks) rather than DOS handles. But the fundamental error, as I see it now, is attempting to make the same block numbers refer to multiple blocks. In other words, the OPEN ... FROM trick is needed—to specify which is the source and which is the destination file—only because the same block numbers are used in both files. Pygmy solves this by giving each file its own range of block numbers. Once that is done, accessing any block in any of the files is straightforward.

Pygmy allows 15 files open at one time. These are all accessible simultaneously at different block numbers. Default files are opened automatically, the defaults can be changed, and additional files can be opened. If you need more than 15 files open at one time, there is supplemental code that shows how to have over 200 files open at one time.

File handling is done relative to the unit number of the file, with such words as UNIT, SETTLE, CHOP, OPEN, and ?CLOSE.

As shipped, Pygmy version 1.3 has two files already installed in units zero and one. These are PYGMY.SCR, which contains all the source code, and YOURFILE.SCR, which contains eight blank screens. These are automatically opened for you and ready to go.

Any time you want to see which files are installed, whether they are open, or what their starting block numbers are, type:

```
.FILES
```

You are not limited to these files! Close them all down with RESET-FILES if you like, and open your own set. If you save that image of Pygmy (i.e., SAVE TST1.COM), whenever you bring up that image (by typing TST1 at the DOS prompt) your custom set of files will be opened automatically and the list of names and starting block numbers will be displayed.

When installing filenames with UNIT, be sure to give each file a different starting block number, so the block numbers do not overlap. It's probably neater to set them up with ascending numbers. For example, unit one could start with block zero, unit two with 300, unit three with 600, etc. All block numbers must be lower than 8192.

Each block in the entire system of open files has its unique number. There is no need to use the F83 OPEN ... FROM CONVEY. To copy the 50 blocks starting at block 17 to the 50 blocks starting at block 300, just say:

```
17 300 50 COPIES
```

The destination can be in the same or a different file, but it is an error if those blocks do not exist. In earlier versions of Pygmy, you could extend a file just by accessing a block past the end of file. In version 1.3, the blocks must already exist. To extend a file, use F9 from within the editor. For copying a single block, you can still use COPY.

FILES keeps track of the highest block number in the file. Neither the editor nor BLOCK will go outside actual file bounds.

HOLES, SETTLE, and CHOP make managing block files more convenient.

Here are some examples:

```
NAMEZ: MYFILE.SCR
```

NAMEZ: is a defining word that creates a Forth word that returns the address of its own name. The string ends in $00 so as to be a suitable "ASCIIZ" filename.

```
FILE-NAME: DATA1
C:\FORTH\WORK\OLD\YOURFILE.SCR
```

FILE-NAME: is a defining word that creates a Forth word that returns the address of the following ASCIIZ string.

```
3000 MYFILE.SCR 7 UNIT
```
Plug the name and starting block number into unit seven.

```
900 DATA1 9 UNIT
```
Plug the name and starting block number into unit nine.

UNIT establishes which starting number and filename go with which unit number. Then, whatever you want to do with a file is done with its unit number (or with the block numbers).

```
7 OPEN
```
Open the file MYFILE.SCR.

```
9 EXISTS?
```
Does file YOURFILE.SCR exist?

```
9 MAKE
```
Create the empty file named YOURFILE.SCR in the C:\FORTH\WORK\OLD\ subdirectory.

```
100 9 MORE
```
Extend that file with 100 blank screens.

```
9 >EOF
```
Position to end of file.

```
9 >BOF
```
Position to beginning of file.

```
3000 3050 SETTLE
```
Make the heavy screens in this range in MYFILE settle to the bottom and the light screens, with only blanks, float to the top.

```
7 CHOP
```
Truncate MYFILE.SCR so that all trailing blank screens are chopped off. Compare this to -TRAILING on strings.

Also available are a few words to make sequential file access easier, such as:

```
FILE-WRITE
( a #bytes unit# -- )

FILE-READ
( a #bytes unit# -- )
```

The Editor

Pygmy has a comfortable, screen-oriented block editor. You can move quickly from screen to screen with the PgDn and PgUp keys, search across screens, insert blank screens, and compress out blank screens. It zips along even on an 8 MHz XT.

Search-across now always goes to the end of file; there is no need to set the ending screen number.

To enter the editor, type n EDIT. To get out of the editor, press the Esc key. When you are in the editor, you can make changes by using the arrow keys to position the cursor and then just typing. Press the Ins key to switch between the insert and the overwrite modes. The backspace key will delete characters to the left of the cursor and the Del key will delete the character the cursor is on. Inserts and deletes only occur on the current line.

The PgUp and PgDn keys are used to move to the previous or next screen. This is delightfully fast. If the cursor is at the beginning of the line already, Home moves to top of screen; otherwise, Home moves to beginning of current line. End moves past the last character on the current line.

The very top line of the screen is a status line that shows the screen number, the filename, and a brief reminder of what some of the function keys do. It also shows an "i" if in the insert mode, and shows a count of the lines in the cut buffer.

F1 repeats a search.

F2 repeats a replace.

F3 sets up a search string and then searches.

F4 sets up a replace string and immediately replaces with it. To repeatedly change CAT to DOG, use F3 to set up CAT, use F4 to set up DOG, and then press F1 F2 F1 F2, etc).

F5 deletes the current line.

F6 joins the line below to the current line at the cursor.

F7 "cuts" the current line to the cut buffer. This does not alter the current line.

F8 "pastes" the oldest line in the cut buffer to the current line on the screen, overlaying the current line. The cut buffer is almost unlimited in size. It can be used to copy and move lines on the same screen or to different screens (even screens in different files). Notice that the count of lines cut (on the status line) changes as you press F7 and F8.

F9 inserts blank screens after the current screen. It is good for opening a file in the middle or for extending a file at the end. As you move screens around and delete them from where they used to be, you may accumulate a number of blank screens. SETTLE (used outside of the editor) is used to let the heavy screens sink to the bottom and let the light screens float to the top. It only affects the range of screens that you specify, e.g.,

```
315 345 SETTLE
```

will re-arrange those screens so that any completely blank screens are at the highest numbers and the non-blank screens are at the lowest numbers. This compresses out blank screens. A related word CHOP will truncate a file by chopping off all trailing blank screens, e.g.,

```
3 CHOP
```

will chop the blank screens off the end of the file whose unit number is 3.

F10 does a search like F1, but across multiple screens.

Esc exits from the editor.

Enter ends the current line, pushing anything to the right onto the following line and pushing the lines below it down.

Summary

I think of Pygmy as a small but complete Forth. It is small enough that you can control it and understand it, yet powerful enough to use for real work. It can be ordered from FIG or downloaded from GEnie (PYGMY13.ZIP) or other BBSs. It is shareware, but no contribution is demanded. The documentation file explains how to obtain a printed glossary, interrupt-driven input serial I/O routines, and double (32-bit) and quad (64-bit) number support.

$ 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

See last issue's editorial for details!

(GEnie, continued.)

guests who have participated in the past are a virtual "Who's Who" of Forth luminaries, and those who have graciously accepted invitations for future conferences promise to be every bit as insightful and interesting. You are cordially invited to join in one of these discussions as well as our informal meetings. I usually host the open Figgy Bar held each Thursday, where no Forth-related subject is taboo. Co-SysOp Leonard Morgenstern is usually host of the Sunday gathering dedicated to learning and technical questions from Forth users.

—Gary Smith
GARY-S on GEnie

(Editorial, continued.)

Forth implementations and associated literature, was acquired recently—lock, stock, and source code—by Glen B. Haydon. He has been involved intimately with those products over the years, as well as with projects like the WISC technology behind Harris Semiconductors' Forth chip and the *All About Forth* dictionary of common Forth usage. Glen looks forward to reinvigorating MVP, and says business has continued as usual since the change of ownership. Roy Martens, the original owner, was for many years closely associated with the Forth Interest Group and is still familiar to many of the membership. We wish all the best to him and his wife Sari.

—Marlin Ouverson
Editor

(Letters, continued.)

member services like *Forth Dimensions* and our mail-order operations." What services are these, exactly?

Is the money going to *Forth Dimensions*? $6.66 per issue is pretty steep for a hobbyist newsletter. (This is not a question of the value of the content. *FD* is worth that much to me. It's just that I can't believe it costs that much to produce!) I'll bet that without the baggage of FIG the magazine could be done for $4 or $5 per issue. If the magazine were allowed to pursue advertising and distribution in a realistic manner, I'll bet it could be even less.

Is the money going to the mail-order operation? If so, why refuse to carry some items on the grounds that they are not profitable enough? I know the mail-order operation *used* to make money a few years ago when Roy Martens was running it.

The FIG convention has essentially evaporated. The efforts I have heard tell about over the years to promote Forth outside the community have never materialized. As a non-Silicon-Valley resident, I really don't see any other FIG benefits (especially now that there is no Chapter Coordinator to promote new chapters, and there are no chapters in my area).

I know that failing to renew will exacerbate any financial problems. If there are problems, the membership deserves to know so that we can participate in the solution. I didn't get to vote for anyone running FIG. For years, the leadership has kept the membership intentionally in the dark as to the running of the organization. As a disenfranchised, lowly FIG rank-and-file member, I can think of only one course of action to follow. I'm voting with my dollars and voting with my feet.

My renewal notice states, "...every member's fee goes to keeping our doors open and the presses rolling." So, what's the problem with proving it? How expensive are the doors you're keeping open? When you publish a complete financial statement in *Forth Dimensions*, I'll restart my membership.

Phil Koopman, Jr.

Thanks for your letter, Phil. It is timely in pointing out concerns relevant to recent or pending actions on the part of the Forth Interest Group's Board and/or Business Group.

First of all, get out your checkbook. FIG President John Hall had scheduled the publication of a financial statement in this issue, but the Board didn't approve its final form by press time. John promised to deliver it for the next issue, and yearly thereafter (see his comment in this issue's "President's Letter"). I hope it provides the kind of information you need in order to understand how the business side of FIG operates.

As to the role of Chapter Coordinator, I agree it could be instrumental in the growth of FIG—and the resultant public awareness of

Forth and Forth resources. And, of course, it should be developed and maintained as a vital, two-way link between FIG "central" and its members.

You may not have seen the FIG organizational chart in our last issue; as it shows, Jack Woehr has been reappointed to the Chapter Coordinator position. We wish him the best—it's a job that could be about as big as he chooses to make it and, like virtue, it is its own(-ly) reward. There is really nothing like a very well run chapter when it comes to generating Forth interest and even developing the local pool of Forth expertise. See the following letter for a case in point!

Thanks for taking the time to stimulate dialogue about your concerns. I hope these comments answer them to some degree.

—Ed.

FIG U.K. Enters Second Decade

Dear Sir,

As England's FIG Chapter (FIG-UK) is celebrating its tenth year, I feel it would be appropriate to give some report of its activities.

With a current membership of over 250 members, FIG-UK is probably the largest—and certainly the most diverse—chapter, with members in every continent. We operate independently from the mother group, publishing our own bimonthly magazine *Forthwrite*, which is of a consistently good quality. FIG-UK has been blessed with an enduring committee which provides the backbone of the group. Special mention must be made of our editor, Gil Filby, who has succeeded in filling every issue of *Forthwrite* with interesting articles and letters. Although the magazine does not have the professional glossy finish of *Forth Dimensions*, it represents remarkable value for money at £10 sterling for six issues of 36 pages or more each.

Meetings have been held on a monthly basis in London since the group started; more recently, Paul Bennett, a member of long standing, has started meetings in the Bristol area, covering the southwest of England. Meetings are generally of an informal nature, ranging from "bring and show" evenings to talks given by members of the group. We also attract outside speakers from time to time, demonstrating Forth hardware and programming products. Indeed, at the moment we are currently in discussion with IBM to host a talk about their Forth-based CAD package "IBMCAD" at their conference center in London. If this comes off, it will be something of a coup for FIG-UK.

As Events and Meetings secretary, I would like to mention that Figgers visiting the U.K. are always welcome at our meetings, and should contact me to arrange details. I will doubtless ask you to speak at the meeting, but should reassure you that it is not compulsory.

FIG-UK is currently involved in the FANSI project, which aims to have an ANSI-compatible Forth available very shortly after the standard is finalized. As this is the group's first major project, we are taking it very slowly, but have high hopes of success. It has generated much excitement, to the extent that it is very likely that the first platform for FANSI will be a 6809-based single-board computer which a member has offered to design specifically for the group.

The first group competition, Forth Programmer '90, was held a few months ago with encouraging results. The challenge was simply to write a single screen of code, with two-thirds of a page of documentation, to demonstrate Forth's compactness. Among the winning entries were a set of string handling words, including constants, literals, concatenation, and other facilities; a demonstration of arithmetic using very large numbers (100 bytes long!); and a cursor-based screen editor with twenty functions, which took some shoehorning to fit into the one-screen limit. We hope to repeat this exercise, or something similar, on an occasional basis.

The group librarian maintains a library of over fifty Forth-related books and reprints, as well as complete sets of various conference proceedings and journals. This is a popular facility, and new purchases are made on a regular basis. Back issues of *Forthwrite* are also available to members for purchase, as are listings.

The group is constantly seeking to recruit new members, to which end we are listed in three of the most popular British computer magazines and, as an incentive, anyone asking for an application form who has an IBM PC or clone is sent a free copy of F-PC. Again, we hope to extend our publicity further, subject to the limitation that this be done at minimum cost. (Available funds are used as much as possible for the benefit of existing members, as a matter of policy.)

Although we remember the halçyon days around 1983—when membership peaked at over 600, primarily due to the lamented Jupiter Ace and the rash of other low-cost home computers—the group has not settled into complacency, but continues to be forward-looking and exciting. It is possible that the developments in stack processors will provide a new crop of potential members, quite different from the hobbyist programmers that once made the bulk of the group, and we will be ready for them.

In conclusion, I am pleased to report that the first decade of FIG-UK has been a good one, and that we expect to prosper in the next decade and well into the next millenium.

Yours faithfully,
Gordon Charlton

We extend our congratulations to all of FIG-UK and commend its long-standing leadership for its commitment, perseverance, and success. It has been all too rare that we have had the opportunity to communicate—independent of the "mother group" indeed! I think there is much to be learned from Gordon's letter, especially by those interested in having a vigorous FIG chapter near them.

Now, FIG-UK, how about sharing those winning entries from Forth Programmer '90 with the readers of Forth Dimensions*? We'd enjoy the chance to get to know you better, and we can think of no better, no more Forthly, way to do so. We hope to hear from you soon!*

—Ed.

A message from the President of the Forth Interest Group

President's Letter

Organization, Issues, Actions, and Explanations

More about the organization, about the problems and issues, about what is happening right now to solve them, how I see the issues, and plans for the future. I started this in the last issue of *FD* and I am going to tell you a little about each in each issue of *FD*. The parts are Organization, Issues, Actions, and Explanations.

It is up to the members to use, and thereby to adopt, a standard.

Organization

Business Group

The Business Group is composed of the officers and others who are interested in the day-to-day business of FIG. We meet in San Jose on the Tuesday before the fourth Saturday, and we are charged with the implementation of the policies established by the Board of Directors. As an example of what is happening, here are some highlights of the business meeting minutes for March, April, and May (full minutes of the Business Group are available upon request).

March 23, 1991

GEnie: Dennis Ruffer represented FIG and the FIG Roundtable at a GEnie Conference for sysops in Maryland.

Publications: A "Yerkes Forth" disk set was added to the "Contributions from the Forth Community" collection of disks.

Operations Update: Marlin Ouverson, besides being the editor of *FD*, will take over the other *FD* tasks previously done by ADC. This will reduce the coordination previously necessary and hopefully will make a more efficient operation.

Theme Issue Contest: Funds were approved to award prizes to authors of the best three articles of a special theme issue on Object-Oriented Programming in a future *Forth Dimensions*.

ADC Reports: There were 944 members to date. The second billing notice was mailed the previous week.

Publicity: (focus topic) Horace Simmons agreed to tabulate the member surveys, about trade publications, that were returned with the renewals.

April 23, 1991

Forth Dimensions: In the President's letter in *FD*, John Hall announced the reappointment of Jack Woehr as the chapter coordinator. John included an organizational chart showing FIG's reporting structure. It shows that Jack will be reporting directly to John

ADC Reports: There were 1381 members to date. ADC was instructed to mail the third billing notice to past members who had not renewed by that date.

FORML Proceedings: Bob Reiling said the *1990 FORML Proceedings* would be ready in May, that it is 450 pages and that euroFORML papers are included.

ANSI Standard: Bill Ragsdale reported upon ANSI progress. Because Bill cannot attend the May meeting, Robert Smith was appointed as the FIG representative.

Publicity: Horace Simmons will prepare a draft of an article on the trade publications survey for a future *FD*.

May 21, 1991

Membership Dues: Membership dues were modified to include a category for students at $24 per year. Other dollar values were discussed, and $24 was felt to be the minimum that FIG needed to cover postage and handling costs.

ADC Reports: There are 1458 members to date. Several people have joined for multiple years. Contributions showed a 100% increase from the prior year.

Issues

Here are additional issues that have been expressed by members who have called me lately.

- FIG needs a hot line where

(Continued on page 35.)

News from the GEnie Forth RoundTable

Best of GEnie

We have spent two issues catching up on recaps of GEnie Forth RoundTable real-time guest conferences. This will close out the current series of these visits with our always interesting and informative guests, with a promise I will not delay so long before the next series.

Guest conferences reviewed here include Frank Sergeant, "Why Pygmy"; George Nicol, "SBC 32"; Bill Ragsdale, "Targeted Applications for Forth"; Glen Haydon, "Forth Common Usage"; and Dick Miller, "To DOS or not to DOS."

If you are following these guest conference recaps for the first time, what you will read are the guests' opening remarks and, occasionally, a few extra comments if they add clarity or depth to the opening remarks.

Frank Sergeant, independent consultant and creator of Pygmy—a minimal Forth kernel based initially on Chuck Moore's cmForth—joined us to discuss his reasons for creating Pygmy and some of his Forth philosophy. 6/21/90

Today, I wrote Pygmy Forth for three reasons.

1. I was unhappy with certain aspects of Laxen and Perry's F83:
   a. The slow and awkward editor
   b. The size and complexity (then what must I think of F-PC?)
2. Certain ideas from cmForth appealed to me very strongly:
   a. PUSH POP (instead of >R R>)
   b. FOR NEXT
   c. no IMMEDIATE word (but it does have immediate words)
   d. simple metacompilation
3. I wanted my Forth to be under my complete control.

Tomorrow, I may have written it for other reasons.

Clarifications:

1. "Pygmy" is short for "Pygmy Forth." It is *not* a new language; it is Forth (as I see it) all the way.
2. Pygmy is *not* public domain; it is shareware. The fee is reasonable (from a minimum of *zero* to a maximum of all you have—your choice).
3. Is Pygmy perfect? Of course not.

George Nicol, president of Silicon Composers. Dr. Nicol discussed his company's SBC32 single-board computer based on the 32-bit Forth engine created at Johns Hopkins Advanced Physics Lab. 7/19/90

Silicon Composers, in business since 1986, specializes in hardware and software using Forth chips such as the 16-bit Harris RTX 2000, the 32-bit SC32 Forth chip, and the NC4016. We make single-board computers for standalone use and for use in PCs as plug-in co-processors.

We also do custom board and software work. Our newest product is the SC/FOX SBC32, a standalone single-board computer for the SC32 Forth chip. The SBC32 is a small Eurocard-size (100 x 160mm) board with up to 512K bytes of SRAM, 128K bytes of EPROM, a 56K baud serial port, and two fifty-pin user application connectors. It also comes with SC/Forth32—an interactive Forth based on the Forth-83 Standard—in EPROM. (See the July/Aug '90 issue of *Forth Dimensions*, inside cover, for additional details). If you have any questions about Silicon Composers or our products, please feel free to ask.

Bill Ragsdale, president of Dorado Systems. Bill discussed his company's approach to "Targeted Applications in Forth." Also discussed were various Forth models and what Bill sees as a major failure by the X3J14 ANS Technical Committee as they construct the new standard. 8/16/90

Hi, all. I'm glad to be here through the magic of GEnie.

We need commercially viable Forth applications that are motivating to the buyer (company) and seller (Forth programmer). This is needed for Forth to be rewarding to all and viable in the long term.

I propose "tailored applications." These are standard and custom software in a product form. The features are:

- quick turnaround
- done in a field the programmer knows well
- have customers under a time pressure

I would look for applications written from an application shell which the Forth programmer customizes. The prices would be $500 to $1,000 in one or two hours, or $5,000–10,000 when it requires one or two days. We do such products at Dorado Systems, in this price range.

Dr. Glen Haydon, author of the newly revised All About Forth *and co-founder of WISC Technology. Glen takes issue with standardization as he discusses how Forth should be defined by practice, not by committee. Glen's topic: "Forth Common Usage." 8/23/90*

No natural language has ever been standardized. CHM *[i.e., Charles H. Moore]* created Forth to overcome the liabilities of an operating system. Using any other

operating system is an albatross. No two hardware systems are the same.

Use Forth to exploit your hardware. Don't cripple your hardware with Forth. Level 0 Forth has about 70 functions. Even these will vary with hardware.

Program for people. Let the computer find and compile what it needs. If you cannot say it in English, don't muddy your thoughts further with muddy code.

In programming each function, include:

1. A clear English functional statement.
2. A precise functional implementation.
3. A set of test vectors.
4. Comments on where you stole the algorithm.

All About Forth (3rd ed., 1990) is a concordance of about 500 functions in common usage. Included are functional definitions and implementations from fig-FORTH, 79-Standard, 83-Standard, MVP-FORTH, F83, F-PC, and original sources where available. Appendices include the complete text from the *fig-FORTH Installation Manual*, the 79-Standard, and the 83-Standard.

Understand your application and your tools. Programming the solution is only ten percent of the job.

Dick Miller is a partner in Miller Microcomputer Services, a cottage industry whose work-at-home programmers develop state-of-the-art microcomputer software at all levels. Dick discusses whether to run Forth on DOS... or not. 9/20/90

Hi friends... This is my first conference via keyboard so be kind, and thanks for your patience.

The topic for tonight's conference, to DOS or not to DOS, echoes some of the Forth philosophy we've been using at MMS since we released the first version of MMSFORTH way back in Spring 1979 in parallel with the early fig-FORTH efforts.

And, while we were waiting six months for an answer to our letter in the FIG mailbox, we rolled our own. Then and since, some of the MMSFORTH programmers are: Tom Dowling, Jim Gerow, Dave Lindbergh, John Rible, Bent Schmidt-Nielsen, and of course myself and Jill Miller. In addition to MMSFORTH, we offer FORTHWRITE, DATAHANDLER-PLUS, FORTHCOM, EXPERT-2, GENERAL LEDGER, 8087 and vector graphics support, plus a wide variety of other utilities and games. Also, custom Forth programming, and standard and special hardware to users worldwide. Sorry for the commercial, but I hope this will open up some interesting question areas. I have listed some areas of interest, but since we still offer a non-DOS version I thought that might be a subject of some general interest.

Transcripts & On-Line Meetings

The full transcript for each of these conferences is available in the GEnie Forth RoundTable Software Library 1. If you have not participated in one of our guest conferences, the list of upcoming dates and guests is posted in advance in the Bulletin Board area, Category 1, Topic 6. The

(Continued on page 30.)

(Smart Comments, from page 7.)

piler words. Next, relocate the dictionary pointer to the desired EPROM address and start compiling the words that will be placed in the EPROM for the application. (The first word in the EPROM must be known. Normally, I make the first word FFFF CONSTANT TRUE.) After loading the application source program, but before saving the binary image of the EPROM code, take the action shown in the last line of Figure Five. The ' TRUE 2- returns the address of the link field for the word TRUE. At the end of the compile, the link field value is the name field address (NFA) of the last of the compiler words. This action changes the entry to the NFA of (, which in effect forgets all of the compiler words from the current dictionary. Therefore, when the compile code is placed into EPROM, interactive dictionary searches will be normal, since the RAM-located compiler words are no longer in the dictionary search thread.

Figure Five. Keeping compiler words out of the EPROM.

```
HEX
FORGET TASK

D000 DP                 ( An area of RAM that won't be used for
                        ( application code during compilation.

\ Load compiler words here.

100 DP !                ( Start of EPROM source code.
' ( NFA ' TRUE 2- !
```

(President, from page 32.)

new members can call and get technical advice.

- *FD* needs technical articles that deal with hands-on construction of Forth projects.
- *FD* needs more articles that are oriented toward new users.
- FIG needs to coordinate with vendors who are willing to promote FIG through their mailings.

Actions and Explanations

Treasurer preparing a review of FIG financial situation for publication in FD.

The final approval of a report style for inclusion in this *FD* had not been completed by the time *FD* had to go to press. The financial statement will be included in the next *FD* and in future July/August issues of *FD*.

Standards—should FIG endorse them?

It is not FIG's place to endorse or reject standards. FIG is the Forth community and has always encouraged use of the current standard adopted by its members. It is up to the members to use, and thereby to adopt, a standard. *Forth Dimensions* will reflect that usage. At the moment, *Forth Dimensions* is preferring the use of the current standard Forth-83. *[Tho' articles written in other Forth dialects are also published. —Ed.]*

Coordination of Forth organizations:

A proposal has been made to investigate a committee that would foster coordination and cooperation with other Forth organizations.

FIG has always supported the other Forth organizations. It is a benefit to our members to have as much Forth exposure as possible. FIG monetarily contributed to the establishment of the Institute for Applied Forth Research and we carry the literature of the Institute and ACM's SIGForth as a benefit to our members. We encouraged the Forth Vendors Group, when it was active. One of the goals of all of these organizations is to increase the public awareness of Forth; toward that goal, we should pool our resources and find a way to coordinate our efforts.

Forth Dimensions

We are currently placing *Forth Dimensions* on the magazine stands of Computer Literacy bookstores in the Silicon Valley. It has been a trial to see how well *FD* would be accepted as a specialized publication among other general publications. Several of us are watching closely to see how well *FD* is presented in the stores and how well it is accepted. So far it looks quite successful. Recently, Computer Literacy commented that *FD* has the cleanest and most professional look of all of its magazines, and has put *FD* in a front position on the magazine rack.

It is time we are willing to try the same in other stores. This still being a trial, if there were bookstores in an area you would be interested in looking after, we would like to have you coordinate placing *FD* in them. FIG being a volunteer organization, this would be a volunteer operation. Contact me for details.

I am always available for comments.

—John Hall
415-535-1294
JDHALL on GEnie

Forth resources & contact information

reSource Listings

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

Forth Interest Group

The Forth Interest Group serves both expert and novice members with its network of chapters, *Forth Dimensions*, 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
John Hall, President
C.H. Ting, Vice-President
Mike Elola, Secretary
Dennis Ruffer, Treasurer
Wil Baden
Jack Brown
David Petty
Dennis Ruffer

Founding Directors
William Ragsdale
Kim Harris
Dave Boulton
Dave Kilbridge

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)

**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 April 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

Forth Modification Laboratory

FORML and EuroFORML

Conference Proceedings

Conference papers from the twelfth annual FORML Conference held November 23-25, 1990 at the Asilomar Conference Center, Pacific Grove, California, U.S.A. and conference papers from the EuroFORML '90 Conference held October 12-14, 1990 at Potters Heron Hotel, Ampfield, Nr Romsey, Hampshire, U.K. 460 pages.

Order from the Forth Interest Group.
FIG member discount available.
See order form inside.

$40.00

FORML

TABLE OF CONTENTS

FORTH IN INDUSTRY
Forth in Industry
A Simple Communications Monitor
6805 Development on a Budget
A Multitasker for the 68HC11
From Below Ground to Outer Space: Forth at RPC

EXPLORING
Using a 3-key Keyboard for Text Ear Training
"Yet Another" Object-Oriented Program in Forth
Controlling Serial Devices

INTO FORTH
Safety Nets for Error Recovery
Generic Stack Operations
Managing Dictionaries as Packages
Event Management in Process Control

WIL'S REVELATIONS
Virtual Rheology
How Many Forks for Deep Spaghetti
How to Uncook Spaghetti
Spaghetti Restructured

EDUCATION AND MATHEMATICS
Forth as the Language of Design in a Microprocessor Systems Design Course
Approximate Rational Arithmetic
A Study of WEE Groups

eFORTH
eForth--The Model, Design and Implementation
A 32-bit 68000 eForth Implementation for the Motorola Educational Computer Board

EuroFORML '90

LANGUAGE AND MATHEMATICS
The Forth + + 'C' Intereface
A Versatile Application Generator for Signal Processing and Data Analysis, using Forth as a Command Interpreter
Floating Point in Industrial Control
Functional Approximation by Chebyshev Series

WORDS AND LANGUAGE
Deferred Language Translation
Proj - a Software Development Project Manager
A Forth-Object Compiler and Its Applications
FEC - Forth Environment Compiler

THE CONSTRUCTION INDUSTRY
The Beauty of Separate Systems
The Inter-Application Execution of Forth Words for Seamless Co-operative Systems
Algebraid Specification of Stack Effects for Forth-Programs
A Database Project for Forth with the Modula-2 Data Model

DEVELOPMENT ENVIRONMENT
A Distributed Forth Environment
Stand Alone In-Circuit Development with Microcontroller "Siemens" SAB 80535
uswFRAME - Forth Real-Time Application Meta-Environment
Philosophy of Control uswFORTH

A TOPIC FOR DISCUSSION
USW Resource Access eXecutive

OBJECTS & OBJECTIVITY
Real-Time Object-Oriented uswFORTH
Little Universe and Object Encapsulation
A Bit of History
Forth and Safety Related Systems
Managing Software Projects Using FORTH

MANY & OFTEN TOGETHER
An Approach to the Programming of Distributed Shared Resources within an Experimental Robotic Workcell
The Bridge Between PLC and Forth
Forth+ + and the Mach-1 Board
Implementation of a Fast Reading Preemptive Multi-Tasking System

Index of FORML Conference Papers — 1980 through 1990

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