$10 Volume XX, Number 1 May 1998 June

FORTH DIMENSIONS

OFFICE NEWS

This issue of *Forth Dimensions* heralds the beginning of the 20th year of the Forth Interest Group. For me, someone who joined this legacy only in the last two years, this milestone seems to signify many years of dedication and innovation on behalf of the programmers and developers who use a language that is used so reliably and silently around the world.

If you've been reading Office News regularly, you know about some of the changes we've been making here at the main office. We may need to make more changes in the coming months. We are acquiring new members at a greater rate than we had been, however, if each member could invite several friends to join the Forth Interest Group, we at the office would be happy to send a complimentary issue of *Forth Dimensions* for their review. March is our major renewal time, and quite a few of you haven't yet renewed at press time. Timely renewal is important to keep *Forth Dimensions* coming to you without interruption.

Cost-wise, we run the Forth Interest Group as lean as we can. FIG is being kept alive by the kind donations of many. Last year, FORTH, Inc. donated a new modem, and the printing of 1000 membership brochures that we use to solicit new members. Taygeta Scientific Inc. donates the space for the forth.org web site and Brandon Yeager's time for system administration of that site. Taygeta also provides Eddy Hamelin's time to answer the phone, to take your orders, and to assist me in any way I need.

This past year, with your donations, we've upgraded the memory on our computer. We had been running with only a 200 Mb hard drive; we now have a 2 Gb external hard drive and have added to the RAM. Our laser printer is beginning to show quite a bit of wear (creaks and groans that don't sound healthy seranade us each time we print); in all likelihood we will need to buy a new one this year. If anyone would like to donate a new laser printer to FIG, please contact us at the main office. We are sincerely grateful to those members who make generous contributions just because they want to. Your extra donations are put to good use.

To be starting our 20th year is quite an accomplishment. The people who have contributed to Forth comprise an impressive list of talented, innovative, and dedicated individuals. They gave life to this organization, and gave you a place to begin your knowledge of Forth or to enhance your ability to use it. The question before us is the one that will be explored at FORML: how does the Forth Interest Group change and grow and continue to meet the needs of the Forth community? We look forward to your continuing support and participation! Hope to see you at FORML...

Together we make the difference!

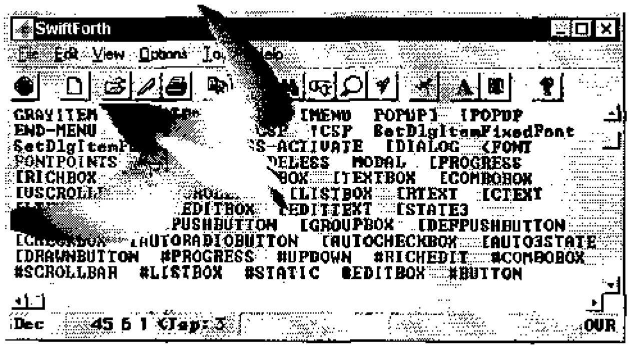
Cheers,

Trace Carter
Administrative Manager
Forth Interest Group
100 Dolores Street, Suite 183
Carmel, CA 93923 USA
voice: 408-373-6784 • fax: 408-373-2845
e-mail: office@forth.org

<u>All-new</u> development environment from FORTH, Inc.

SwiftForth™

for Windows 95/98 and Windows NT



```
GRAYITEM                  [MENU  POPUP]  [POPUP
END-MENU               CSP  !CSP  SetDlgItemFixedFont
SetDlgItemF          S-ACTIVATE  [DIALOG  <FONT
FONTPOINTS            DELESS  MODAL  [PROGRESS
[RICHBOX               BOX  [TEXTBOX  [COMBOBOX
[USCROLL                [LISTBOX  [RTEXT  [CTEXT
              EDITBOX  [EDITTEXT  [STATE3
            PUSHBUTTON  [GROUPBOX  [DEFPUSHBUTTON
[CHECKBOX  [AUTORADIOBUTTON  [AUTOCHECKBOX  [AUTO3STATE
[DRAWNBUTTON  #PROGRESS  #UPDOWN  #RICHEDIT  #COMBOBOX
#SCROLLBAR  #LISTBOX  #STATIC  #EDITBOX  #BUTTON

Dec    45 6 1 <Top: 3                                OUR
```

- Super-efficient implementation for *speed* (32-bit subroutine-threaded, direct code expansion)
- Full GUI advantages (like drag-and-drop editing; hypertext source browsing; visual stack, watchpoints, and memory windows) but retains traditional command-line control and tools
- Complies with ANS Forth, including most wordsets
- Easy to add DLLs and to call DLL functions
- DDE client services for inter-application communication
- Files *and* blocks supported
- Simple creation of windows, menus, dialogs, etc. — no third-party tools needed
- Flexible, extensible access to system callbacks and messages, and exception handler

FORTH, Inc.
111 N. Sepulveda Blvd., #300
Manhattan Beach, CA 90266-6847
800.55.FORTH • 310.372.8493 • FAX 310.318.7130
forthsales@forth.com • www.forth.com

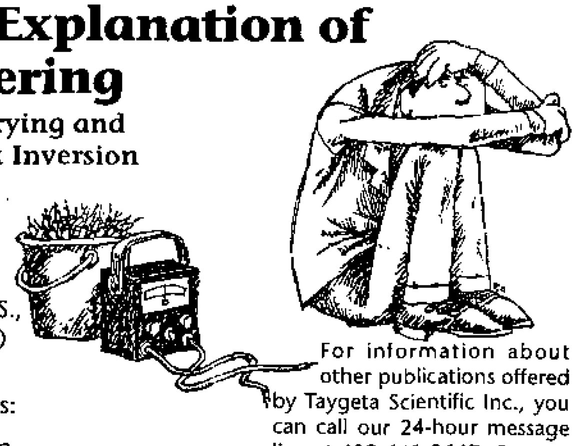
Call today for data sheet or visit our website!

This classic is no longer out of print!

Poor Man's Explanation of Kalman Filtering

or, How I Stopped Worrying and Learned to Love Matrix Inversion

by Roger M. du Plessis



$19.95 plus shipping and handling (2.75 for surface U.S., 4.50 for surface international)

You can order in several ways:

e-mail: kalman@taygeta.com
fax: 408-641-0647
voice: 408-641-0645
mail: send your check or money order in U.S. dollars to:

For information about other publications offered by Taygeta Scientific Inc., you can call our 24-hour message line at 408-641-0647. *For your convenience, we accept MasterCard and VISA.*

Taygeta Scientific Inc. • 1340 Munras Avenue, Ste. 314 • Monterey, CA 93940

DEPARTMENTS

EDITORIAL

Change is the Constant

In an editor's life, there is one constant: the search for good material to publish that will satisfy the range of tastes evinced by a publication's readers. That's why you find me frequently reminding you to write for us. *We are actively seeking* new articles, announcements, letters to the editor, and even an occasional columnist. Please consider sharing your thoughts and experiences (and, yes, your code) with your fellow Forth users.

• • •

Change can be disconcerting, but it can also bring new opportunities. Over the years, numerous discussions have taken place about the various resources of the world-wide Forth community and about how better cooperation and coordination can provide additional leverage and greater opportunities for all concerned.

In that vein, below is an announcement from the leaders of the Forth Interest Group and The Forth Institute. We are looking forward to the results of this news. In fact, the first tangible results will appear in these pages in the next issue—stay tuned, and let us (and our authors) know your reactions.

—Marlin Ouverson, Editor

"For several years, the *Journal of Forth Applications and Research* has been the pre-eminent location for refereed papers on Forth technology and its application. Beginning with Volume VII of the Journal, *JFAR* has become electronic and can be found on the Web at www.jfar.org. Its new editor is Dr. Peter J. Knaggs, of the Bournemouth University in the U.K. As an additional service to the Forth community, selected refereed papers from *JFAR* will now appear in a special section of each issue of *Forth Dimensions*. These papers will represent both the currently electronically published volume and significant papers from previous volumes.

"It is our hope that, through *Forth Dimensions*, these papers will find a new and larger audience. By expanding *Forth Dimensions* to include a section for the hard copy publication of peer-reviewed papers from *JFAR*, the Journal section of *Forth Dimensions* can now provide an important means of getting important papers about Forth (particularly from the academic community) widely disseminated.

"We hope you will be inspired by these papers to explore your own work in detail, in both *Forth Dimensions* and in *JFAR*."

Skip Carter
President, FIG

Larry Forsley
The Forth Institute

Would you like to brush up on your German and, at the same time, get first-hand information about the activities of your Forth friends in Germany?

Become a member of the
German Forth Society
("Deutsche Forth-Gesellschaft")

80 DM (50 US-$) per year
or 32 DM (20 US-$) for students or retirees

Read about programs, projects, vendors, and our annual conventions in the quarterly issues of *Vierte Dimension*. For more information, please contact:

Fred Behringer
Planegger Strasse 24
81241 Muenchen
Germany
E-mail: behringe@mathematik.tu-muenchen.de

Forth Dimensions
Volume XX, Number 1
May 1998 June

Published by the
Forth Interest Group

Editor
Marlin Ouverson

Circulation/Order Desk
Trace Carter

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

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

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

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

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

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

NEW PRODUCT ANNOUNCEMENT

340 Mbyte Continuous Fast Storage for Pocket Data Logger Module

Victor, NY.—The TDS2020D is a Forth-based, pocket-sized datalogger module which now provides continuous fast data collection to PCMCIA cards, without any break caused by transfer of data to hard disk or flash-ATA card. It will work from a small battery for months, storing data on Windows-, DOS- or OS/2-formatted cards.

Fast data-logging under interrupt into PCMCIA disks or flash-ATA cards is achieved with a double data cache. One cache is used for data collection under interrupt, while data in the other buffer is being transferred to the PCMCIA card by the foreground routine (or another task, if the multitasker is installed). Data collection speeds of over 100,000 bytes per second can be achieved.

High-level, ANS Forth data-logging programs provided can be used immediately, but are customizable for individual applications. By taking several samples on each interrupt, data rates over 100,000 bytes of ten-bit A-to-D data per second can be achieved. The rate at which data can be collected into one half of the buffer is ultimately limited by the time needed for the foreground program to push the alternate half of the data to disk.

For use anywhere large amounts of portable data have to be obtained, the TDS2020D stores data in PC-file format onto PCMCIA cards for subsequent analysis on a PC.

Typical current for computer, adapter and hard disk is 350µA standby, 30mA operating. For example, 24 bytes of digital, analog, and time information logged every minute will cause the hard disk to power up for only five seconds every two weeks.

Returning the PCMCIA disk from the field, the storage device can be put into a PC's PCMCIA slot and, without any special software, data can be copied to a PC file.

Saelig Company LLC
1193 Moseley Road
Victor, NY 14564
716-425-3753; fax 716-425-3835
saelig@aol.com • www.saelig.com

ANS FORTH UPDATE

Excerpts from a letter by the chair of the ANS Forth Technical Committee (TC)—According to rules governing ANSI standards, four years after a standard is published, its TC must vote to "reconsider, reaffirm, or withdraw" it. As ANS Forth was published in 1994, this is the year. Note that if we fail to act, it will be withdrawn for us....

1. The TC shall get a letter ballot to vote to "reconsider" the standard. If this fails, the other two choices are to "reaffirm" or "withdraw," which would be a subject of a second ballot in that case. If it passes, we'll propose a first meeting to coincide with the 20th FORML Conference at Asilomar, California, on the weekend of Nov. 20–22 and a second meeting to coincide with the next Rochester Conference (to be held somewhere other than Rochester).
2. If we vote to "reconsider," we will do so with an agenda limited to the following items: (a) Withdrawal of "obsolescent words," (b) Ratification of "clarifications" passed since publication of the Standard, (c) Support for embedded and ROMable systems, and (d) Support for internationalization and extended character sets.
3. The above agenda may have additional topics added by a 2/3 vote of the membership.
4. People have requested consideration of additional topics, such as graphics and multitasking. However, consistent "common practice," or proposals reflective of existing "common practice," seems to be in short supply. The current SD-2 provides a mechanism for such issues: a "Technical Group" (TG), which is a sub-group of the TC given a specific mission, whose product is a "Technical Report." A TR doesn't have the official standing of a Standard...but can serve as a basis for implementations...until the technology has matured sufficiently for a standard.
6. We propose to require that all proposals be submitted electronically, and meet [certain] criteria...

...The present annual fee for TC members and observers is $300. There was a question of whether an additional $300 would be assessed for international representation; the ruling is that, since we do not have an active ISO liaison or working group, it is not applicable.

The $300 fee pays for a principle and one alternate....Membership of TGs is not limited to TC members; however, TG members who aren't TC members must pay $300/yr.

Until our next meeting, you can become a full member by voting in two successive letter ballots (your vote on the second one counts). For this reason, I think it's a good idea to submit letter ballots on the issues from the organizational meeting. When we become active, it still takes two meetings, although NCITS [National Committee for Information Technology Standards, pronounced "insights"; formerly X3.] is considering liberalizing this, since many groups meet infrequently (as we propose)... You can *lose* your membership by failing to pay your fees, by failing to respond to 80% of the letter ballots in a calendar quarter, or by failing to attend three successive meetings. Both the attendance and letter ballot voting requirements apply; you may not skip meetings and just vote electronically! ....There will be no IEEE fee waivers.

If you are not presently a TC member and wish to be included in the letter ballots in order to become a member, please notify me (erather@forth.com) and Greg Bailey (greg@athena.com).

Membership is open to any person or organization who is "materially affected" by the subject matter (by their own definition). It is not limited to U.S. members, so long as we are strictly an ANSI group....A consultant who wishes to be a member must not receive primary funding (e.g., >50%) from any voting member.

—*Elizabeth Rather*

eForth for Java

eForth for Java, or *jeForth*, is a high-level Java implementation of eForth for the Java Virtual Machine (JVM). It runs as a console-style applet which can be opened in a Java-enabled web browser such as Netscape Navigator or Microsoft Internet Explorer. This version of eForth has been extended to provide features common in other Forth systems, such as `FORGET`, `CREATE ... DOES>`, `DO ... LOOP`, and simulated `BLOCK` I/O.

jeForth can open new opportunities for promoting and teaching Forth to a wide audience over the Internet. The author intends to freely distribute this system, including its source code, to non-commercial users and organizations such as the Forth Interest Group.

Purpose

Another Forth written in Java? Yes, and I think this version is substantially different and useful in its intended niche. Starting back in 1995 with the introduction of Java technology, my vision was to create a simple Forth to demonstrate and promote over the Internet. The popular web browsers support Java applets, so it occurred to me that many people could easily try a "live" Forth on the Internet if it is built from Java. Furthermore, a Forth applet can be surrounded with and linked to tutorial texts, making it easy for a student of Forth to experiment while referring to lessons online. Lowered hurdles to learning about Forth may help revive widespread interest in our favorite language and development tool!

A secondary and more personal purpose for this project was to learn more about Java. I have several years of experience with C++, and learned that Java is another object-oriented language with similar syntax and flavor. At least from an academic point of view, I liked some Java features that make it easier and more reliable to use than C++, but I knew that I would not truly understand its benefits and limitations unless I did a substantial project with it. And like many Forth enthusiasts, I could not resist the urge to implement Forth on an emerging platform. So a Forth applet in Java seemed like a fun and worthwhile exercise. It has been, all in all, but Java's security restrictions and problems with early Java tools have made design and development less joyful than I had hoped.

A third possible purpose for a Java-based Forth is as a general internet/intranet application development language. The current version of jeForth is too limited for serious internet application development, lacking such services as HTTP GET and PUT requests, HTML forms processing, Java Bean integration, Java Database Connectivity, and Remote Method Invocation. I am planning another Forth, more suitable for commercial internet purposes, which may include the features listed above and others that web developers would want. Such a commercial package will have a different name and license restrictions. But for non-commercial uses, jeForth is released as an open system, so you can improve and extend it as you wish.

Approach

I considered various Forth models, including some written in C since Java is very similar to it. But the C-based Forth systems are fairly large and I do not have much experience with them. eForth is small and I understand it well, so I chose it to be the foundation. The initial eForth model was implemented for the 8086 processor family by Bill Muench and Dr. C.H. Ting using the MASM macro assembler. Other implementations for various platforms have since been developed, some using MASM with additional macros or hand assembly of the cross-platform portions, some in other languages. eForth has a small number of kernel words in native code, then provides the rest of the Forth environment as high-level "colon" words.

MASM source is not a good match for Java, so I ended up using Java to develop something between a cross-compiler and a Forth metacompiler, which is described below. The MASM source was ported to this special syntax. The result is not very elegant, but it works. When you open the jeForth applet, it takes a moment to actually build the high-level Forth words in memory using these metacompiling routines.

When running, jeForth is mostly in this high-level Forth. The kernel words and Forth VM are implemented in high-level Java code, which itself runs on the Java VM, whose virtual machine instruction *bytecodes* are interpreted (like Forth's inner interpreter) or compiled into native code, possibly with a "Just-In-Time" compiler. These multiple software layers inhibit performance, but should suffice for demonstration and training purposes. Furthermore, by providing most of the Forth system as colon definitions, a student can more easily get "under the hood" to understand how Forth systems are designed and operate.

eForth Virtual Machine

The eForth VM is a set of Java data structures and routines (*methods* in Java-speak) organized into the `ForthMachine` class. The eForth memory is 65536 words of 32 bits each, plus another 8192 words for `BLOCK` I/O buffers, stored in an array of Java 32-bit integers named "`m`." To simplify porting, the memory organization was kept very close to the original eForth except for the `USER` variables, which are stored in low memory. All eForth data, including parameter and return stacks, code, name dictionary entries, and user variables are located in `m`. Figure One shows a memory map.

Forth usually assumes fairly free access to all memory; Java does not. In the name of security and program robustness,

Michael A. Losh • Detroit, Michigan
mlosh@tir.com

The author is a Microsoft Certified Trainer for Windows programming with the Win32 API, Visual C++, and MFC, and is the director of software consulting at American Systems Technology, Inc.

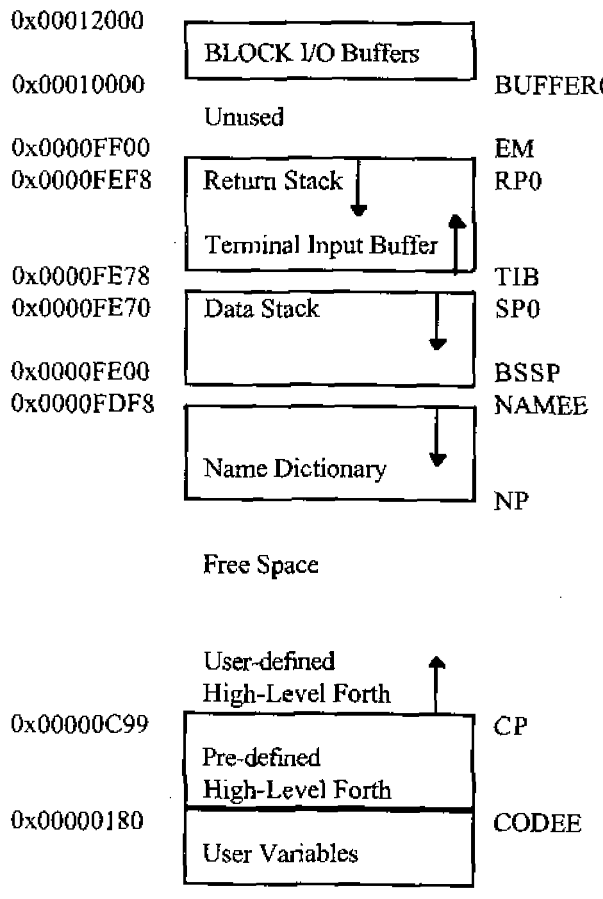


Figure One. Memory map

Java does not offer pointers like C and C++. Mark Roulo did a good job describing these issues in his article on Misty Beach Forth in the November/December 1997 issue of *Forth Dimensions*. From what I read, he used a more sophisticated approach using multiple data types and Java references. I simply allocate a big array and use indices as pointers. After all, what more is a "real" memory address than an index to a RAM location? My approach was easy to use, but undoubtedly sacrifices speed.

I also sacrifice some space, because this Forth does not address bytes! Each eight-bit character is stored in a 32-bit cell. Of course, this scheme wastes 75% of its bits, but it greatly simplifies the VM to have everything treated as 32-bit cells. Java strings can be converted to the 32-bit characters easily, as needed, using the routines `ForthMachine.strlit` and `ForthMachine.makeString`. The consistency of cells allowed me to optimize away some of the alignment instructions, as well as replacing `CELL+` and `CELL-` with fast `1+` and `1-` primitives. `CELL+` and `CELL-` can still be used by applications, of course. The current version uses 2841 cells (11,364 bytes) for code tokens, and 2158 cells (8632 bytes) for the Name Dictionary, so the actual waste is not excessive, considering that platforms running Java typically have far more memory than this. The lack of byte addressing may put off some people but, in a way, addressing eight-bit bytes is somewhat archaic in this day and age of cheap and plentiful 32-bit processors. jeForth proves that Forth (or any programming system) does not need byte addressing.

jeForth supports token threading. The eForth kernel is a small set of primitive words that are implemented in native code, which in this case is high-level Java Development Kit (JDK) 1.0 code and Application Programming Interface (API) calls. The 32-bit integer tokens for these primitive routines have bit 17 set (0x00020000). High-level words, such as colon words, use their code "address" (array index in `m`) as a token. Note that all valid code addresses will be smaller than the value implied by the primitive bit. During execution of Forth words, the inner interpreter (found in the function `ForthMachine.run` of the `ForthMachine` object class, shown below) gets the next instruction token. If the token is a primitive routine, the function `ForthMachine.doPrim` is called. This function is really a large `CASE` structure with the Java instructions for each primitive. Otherwise, if the token indicates a high-level word, the inner interpreter nests into that routine, first saving a return address on the return stack. Token threading works well for a virtual machine implemented in a high-level language. A few of the eForth kernel words had to be redesigned, because they assume direct threading and direct execution of native CPU instructions.

[See Listing One]

A few primitives were added to the kernel: `UM*` and `UM/MOD` for math, `1+` and `1-` for fast increments and decrements, `PICK` and `DEPTH` for working with the stack, a few diagnostic control routines, a primitive to set up the `USER` variables area, and `BLOCK` I/O primitives for reading and writing the `BLOCK` file. Some of these were required by the eForth VM; others were added for performance or feature improvements.

The eForth Name Dictionary was implemented in a way very similar to other eForth systems, but the name length and special flags are stored in a 32-bit cell instead of a single byte, and each character of a word's name is stored in a separate 32-bit cell.

Metacompiler

Porting to Java, I had to replace some of the functionality of MASM. In some ways, I went beyond what MASM does. I defined `ForthMachine.header` to set up a header in the name area of the dictionary. `ForthMachine.primitive` is a special routine to set up a special header for a primitive word. There are routines to mark words as `IMMEDIATE` or `COMPILE-ONLY`. For putting code and data of different kinds into the code area of the dictionary, I made `ForthMachine.code` and `ForthMachine.literal`. I had to implement dictionary searching for `ForthMachine.call` so I could compile calls to other high-level Forth words. The Java dictionary searching functions are only used for building the jeForth dictionary and code image at initialization, not for later colon-compiling the user's words.

To metacompile control structures, I implemented several Java routines with names like `ForthMachine.compIF`, and `ForthMachine.compTHEN`. Here I departed quite a bit from the MASM coding style because it would be difficult to implement MASM-like labels for branch targets. I went with the Forth approach instead! Here is a sample of the metacompiler "code" for the high level word `FORGET` [see Listing Two].

A few of the more complex eForth words were a little difficult to convert to this style, but the "eForth in Forth" listing published by Dr. Ting in the *eForth Implementation Guide* helped, as did a temporary tool I built to display a word's raw code.

Listing One

```
public void run()
{
    try
    {
        // Inner interpreter loop
        while (!bGone)
        {
            try
            {
                int inst = m[ ip++] ;      // Read current instruction,
                                           // advance instr. pointer
                if ((m[ TRACING] & SHOWING) != 0)
                {
                    inform(inst);          // Report state if tracing
                }
                if (inst > PRIMITIVE)      // Check for primitive bit
                {
                    // Strip off primitive instruction bit,
                    // execute primitive
                    doPrim(inst - PRIMITIVE);
                }
                else                       // Nest into colon word
                {
                    rp--;
                    m[ rp] = ip;           // save ip for return
                    ip = inst;             // "inst" is word's address
                }
            }
            catch(ArrayIndexOutOfBoundsException e)
            {
                app.print(" address");
                // Duplicate Forth THROW
                rp = m[ HANDL] ;
                m[ HANDL] = m[ rp] ;
                rp++;
                sp = m[ rp] ;
                rp++;
                pop();
                push(NULLSTR);  // blank error string
                doPrim(EXIT);
            }
        } // end while
    }
    catch(IOException e)
    {
        app.showStatus("Runtime Exception: " + e.toString());
        return;
    }
}

public void doPrim(int inst) throws IOException
{
    int  a, b, c, i, n;      // temporary integers
    char ch;                 // temporary character

    switch (inst)
    {
        // BYE       ( -- )    Exit eForth.
        case BYE:   bGone = true;   break;
    // ?RX      ( -- c T | F )
    // Return input character and true, or a false if no input.
    case QRX   :
        yield();
        if (keys() > 0)
        {
            push(dequeueKey());
            push(TRUE);
        }
        else
        {
            push(FALSE);
        }
        break;

        // TX!      ( c -- )
        // Send character c to the output device.
        case TXSTO :
            ch = (char)(pop() & 255);
            app.emitChar(ch);
            yield();
            break;

        // !IO      ( -- )  Initialize the serial I/O devices.
        case STOIO: break;

        //  doLIT   ( -- w )  Push an inline literal.
        // ip points to inline value,
        // push it on stack and advance ip
        case DOLIT: push(m[ ip++] );  break;

        // EXIT ( -- )    Terminate a colon definition.
        // resume instruction at saved address
        case EXIT:  ip = m[ rp] ; rp++;   break;

        // EXECUTE  ( ca -- ) Execute the word at ca.
        case EXECU:
            rp--;
            m[ rp] = ip;         // save current ip
            i = pop();
            if (i>PRIMITIVE)
            {
                // Execute Primitive
                doPrim(i - PRIMITIVE);
            }
            else
            {   // point to new code so it will execute next
                ip = i;
            }
            break;
```

(The rest is omitted.)

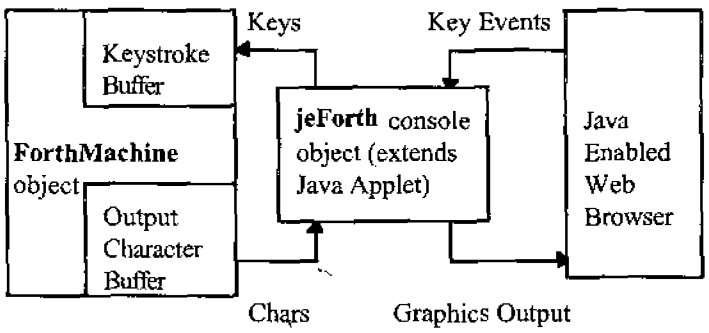
Listing Two

```
//  FORGET   ( "name"  -- )
//  Forgets all the recent words back to and including
//  the named word in the CURRENT vocabulary.
header("FORGET");    literal(32); call("WORD");
                     literal(CRRNT); code(AT);
                     call("find"); code(DUPP);
                     compIF();
                         code(SWAP); literal(CP);
                         code(STORE);
                         code(DEC); // <-- opt for CELL-
                         code(AT); code(DUPP);
                         literal(LAST); code(STORE);
                         // opt below for: 2 CELLS -
                         code(DEC); code(DEC);
                         literal(NP); code(STORE);
                         call("OVERT");
                     compELSE();
                         code(DROP); code(DROP);
                         literal(-1); call("abort\"");
                         strlit("which ");
                     compTHEN();
                     code(EXIT);
```

Console and I/O

The jeForth applet class provides a simple console-style user interface that can be opened in a web browser. In its current form, it presents a 20 row by 72 column display with a simple underline cursor. You may type Forth commands and use the backspace key, but other cursor movement keys are not (yet) supported. The display will automatically scroll up as needed.

The Forth VM and the console communicate through two circular queues stored in the ForthMachine object: one for keyboard characters, and one for display characters. Since the eForth VM and the console run under different Java threads, the queues provide a synchronized interface between them. The current console is not very fast, but optimizations have been made to avoid some of the window repainting. The overall effect of using the console is like working with a Forth system over a serial link at a modest baud rate. The overall architecture is shown in Figure Two.



eForth Extensions

As a demonstration and teaching system, I wanted jeForth to have many features missing from regular eForth but common to other Forth systems. I wanted the system to be compatible with Leo Brodie's excellent introductory book *Starting Forth*. To achieve these goals, I have added several features to the eForth foundation. Compliance with American National Standard (ANS) Forth would also be nice, but it has not been a priority for me. Wonyong Koh has developed a variation of eForth called hForth, for 8086 and other processors, that is ANS compliant; perhaps some of that system can be incorporated into jeForth at a later date.

To help teach new Forth programmers about BLOCK storage, I implemented a "simulated" BLOCK wordset. The simulation comes from the fact that your changes to any block are not permanent: once you leave or restart the applet, the BLOCKs revert to whatever source code is defined in its web page. This rather significant limitation comes from Java's inherent security philosophy, sometimes called the *Java sandbox*. Untrusted Java applets are not allowed to read or write to any local files on the client computer and cannot access other potentially sensitive resources, such as the system clipboard. The only way to load or save information is to use the applet's server. Since I did not want to get into serious server-side development at this time, I have left it out. In the initial release of jeForth, up to ten source blocks can be defined in the hosting web page's applet section, using PARAM tags. For example, here is a section of the HTML web page demonstrating release 1.00 of jeForth [see Listing Three].

During the applet initialization, this code will be read into a large character array that represents the user's BLOCK file. The user can then view the current INDEX, LIST individual BLOCKs, LOAD them, and edit them using most of the *Starting Forth*'s "Find - Put" line-editing commands. The fact that permanent copies of your Forth code changes cannot be made is not crippling for a demonstration and teaching environment, but is unfortunate. If jeForth can send GET and PUT (or POST) HTTP requests, then the server could store the user's BLOCKs. I hope this can be attempted in the future.

In the example HTML for jeForth, you will notice the parameter tag with a name of "boot." Once jeForth reads its BLOCK parameter statements, it will load and execute any boot string you provide. This allows the web page to be self-booting into your application. The block and boot parameter tags give a web developer a means to build a rudimentary but fully interactive web site. Also note that placing single or double quotation marks in the tag source statements is tricky. In HTML, you can place single quotes in a double-quoted string, and vice versa, but it is difficult to do both. In the line with "block1.1" name, you can see a "&#39;" this is HTML's way of inserting a specific ISO-encoded character—in this case, the single quote.

Some other obvious additions to eForth were DO ... LOOP and CREATE ... DOES> words. DO and LOOP have been Forth's traditional indexed looping words. Like Frank Sergeant's Pygmy Forth, eForth offers the simpler FOR and NEXT for indexed looping, based on the influence of Chuck Moore's cmForth. But I thought that providing the DO ... LOOP words is important for a teaching system. Examples from *Starting Forth* and other tutorials using DO ... LOOP now work in jeForth.

Regular eForth provides the CREATE word for defining

Listing Three

```
<APPLET code="jeForth.class" width=540 height=310>

<param name=block1 value="">
<param name=block1.0  value='( Chapter 1, "Fundamental Forth" )                        '>
<param name=block1.1  value='( Sample Forth from Leo Brodie&#39;s "Starting Forth" book )    '>
<param name=block1.2  value="                                                          ">
<param name=block1.3  value="( LARGE LETTER-F )                                        ">
<param name=block1.4  value=": STAR     42 EMIT ;                                      ">
<param name=block1.5  value=": STARS    1- FOR  STAR  NEXT ;                           ">
<param name=block1.6  value=": MARGIN   CR 30 SPACES ;                                 ">
<param name=block1.7  value=": BLIP     MARGIN STAR ;                                  ">
<param name=block1.8  value=": BAR      MARGIN 5 STARS ;                               ">
<param name=block1.9  value=": F        BAR BLIP BAR BLIP BLIP CR ;                    ">
<param name=block1.10 value="                                                          ">
<param name=block1.11 value=': GREET   ."  Hello, I speak Forth " ;                    '>
<param name=block1.12 value=": FOUR-MORE   4 + ;                                       ">
<param name=block1.13 value=": SIDES   STAR  SPACES  STAR ;                            ">
<param name=block1.14 value=": NOTHING  ( do nothing ) ;                               ">
<param name=block1.15 value="                                                          ">
...
<param name=boot value="CR 1 10 INDEX">

</APPLET>
```

named areas in memory, but does not provide the DOES> command to associate special run-time behavior with the name. Since extensibility and defining words are important concepts for students of Forth, I added DOES> to jeForth.

Another command conspicuous in its absence from eForth is FORGET, used to remove recent words from the dictionary. I have added it to jeForth, but it only searches the CURRENT vocabulary. I have not added any of the common vocabulary creation and management words, but these words can be added later.

Experienced programmers can see in their minds what is going on in the Forth VM in terms of stack operations, subroutine nesting, and so on. If less experienced users of Forth can *actually* see these actions, they too will quickly learn to visualize them. To provide this visibility, I have added two words: TRACE and STEP. TRACE looks up the name of the word following it, then executes that word showing a display of the important VM parameters between each token execution. The display includes the top stack item, top return stack item, and the current instruction pointer and the name of the token to which it points. If there is more than one item on the parameter stack, up to three more items in each stack are shown along with the depths of the stacks. STEP does the same thing, but waits for a user keystroke between each token execution. Either will revert to "quiet" execution if the user presses the escape key. Here is a sample output from applying TRACE to execution of the word PAD [see Listing Four].

Another frustrating area for new Forth programmers is memory access, since it is so easy to get stack arguments in the wrong order or otherwise use an invalid address. For better performance and flexibility, typical Forth systems do not try to detect this situation and will crash when a bad address is used. I have used a free feature of Java to detect and warn the user of the problem. Java "throws" exceptions which can be "caught" in a nested way in Java source code.

One type of exception is for "invalid array index." The inner interpreter code (see above) catches this exception, then executes the equivalent to the eForth THROW code, printing an error message and restoring the stacks. The impact of this feature is negligible on performance because Java will do these checks anyway, so we might as well take advantage of it. The user sees an "address ?" warning when executing something like "-1 @", or something that seems perfectly reasonable like " ' SWAP 20 DUMP". By the way, the DUMP example fails because SWAP is a kernel primitive and its token is not a valid address. The code for SWAP is written in the doPrim( ) function, which is completely outside the memory "viewable" from jeForth execution. To see the code for DUP, look at the Java source code!

Listing Four

```
TRACE PAD[enter]
stack: ---    return:      2245   ip:    2245   : PAD
stack: ---    return:      2245   ip:    718    : HERE
stack: ---    return:      718    ip:    714    doLIT
stack: 34     return:      718    ip:    716    @
stack: 3225   return:      718    ip:    717    EXIT
stack: 3225   return:      2245   ip:    719    doLIT
stack: 80     return:      2245   ip:    721    : +
(2)           3225   (10)         2258
stack: 80     return:      721    ip:    513    UM+
(2)           3225   (10)         2245
stack: 0      return:      721    ip:    514    DROP
(2)           3305   (10)         2245
stack: 3305   return:      721    ip:    515    EXIT
stack: 3305   return:      2245   ip:    722    EXIT          ok
```

Performance

As discussed earlier, jeForth is not designed for performance. Early versions of it were very slow, hampered by improper use of Java threads and the `yield( )` function. Early display routines were also inefficient, causing a lot of unneeded repainting of the applet window. These basic problems have been fixed, but there is room for a lot of improvement. To give you some idea of the relative performance of jeForth, I have applied the following benchmarks used by Mark Roulo:

Stack test:

```
: INNER  10000 0 DO 34 DROP LOOP ;
: OUTER  10000 0 DO INNER LOOP ;
```

Variables test:

```
VARIABLE TEMP
: INNER  10000 0 DO 34 TEMP ! LOOP ;
: OUTER  10000 0 DO INNER LOOP ;
```

However, for the slower jeForth, I only ran 1000 loops in the `OUTER` word and multiplied the result time by 10. The figures below may not be extremely accurate. The test system is a Gateway 2000 486 running at 66 MHz with 24 Mb of RAM. The software environment includes Microsoft Windows 95 and Internet Explorer 3.0. The word `OUTER` was timed for two native code Forth systems for Windows: LMI WinForth (16-bit Windows 3.1 code) and the public domain Win32Forth, as well as the Java Forth systems Misty Beach Forth and jeForth:

Forth System	Stack test (seconds)	Variable test (seconds)
LMI WinForth 1.01 (optimized)	8	12
LMI WinForth 1.01	70	107
Win32Forth 3.5 build 41	65	128
Misty Beach Forth V0.30 on IE 3.0 (Java)	199	350
jeForth V0.94 on IE 3.0 (Java)	1076	1891

As you can see, jeForth is about five times slower than Misty Beach Forth on the same hardware, and much slower than either native code Forth systems for Windows. My results for Misty Beach Forth were not as good as those obtained by its author, who tested on a Pentium PC and Windows NT 4.0. Offsetting the slower execution speed of jeForth somewhat is its small size. The two Java bytecode .class files for jeForth total about 36 Kb, which is much smaller than Misty Beach Forth and quicker to download. For a demonstration and teaching tool, fast downloads are probably more important than execution speed. Any serious student of Forth will move on to better performing native implementations, unless doing web-oriented development.

Development Notes

I started with the Sun Java Development Kit (JDK 1.0) to compile Java code, and test it. The very first versions of jeForth ran as a command-line text application and could not run in a browser. Once I based the code on the Applet class, I started using the Sun Applet Viewer and Netscape Navigator version 3.0 for testing. Soon, I felt limited by the constraints of the JDK command-line compiler, so I switched to Microsoft Visual J++ 1.0. This package compiled Java quickly and had the nice graphical debugger with which I was familiar from Visual C++. However, each time I ran the applet, J++ launched Microsoft Internet Explorer 2.x, which took a very long time to come up on my 486, perhaps one or two minutes! I speculate that IE 2.x was waiting for some TCP/IP timeout before opening the local HTML and Java .CLASS files. How frustrating! Later, Microsoft Visual J++ 1.1 came out, which works with IE 3.0 and opens the browser and applet in a reasonable amount of time, although still very slow by Forth-interaction standards.

Testing a Forth that contains a lot of high-level definitions can be a challenge. You can use the debugging tools provided by your assembler or compiler, but those tools will not understand the Forth data structures and run-time environment very well. And if the Forth outer (command-line) interpreter is not working, interaction tends to be quite limited. During this phase, I developed test scaffolding to place different strings of jeForth tokens in memory and execute them. Later I added several debugging features to the Java `ForthMachine` class and to eForth itself. Initially these routines sent output to the standard output stream. Only recently did I re-engineer them to output text into the applet window and add the interactive `TRACE` and `STEP` commands.

I have used the following books to learn about Java:
The Java Programming Language, by Ken Arnold and James Gosling.
Java in a Nutshell, by David Flanagan.

The following books from Offete Enterprises describe eForth:
eForth Implementation Guide, Dr. C.H. Ting
eForth and Zen, Dr. C.H. Ting

Availability

The web pages for jeForth beta versions up to 0.95 have been hosted by Jack Woehr, the Author of *Forth: The New Model*, at his Rocky Cost Free Board:
http://www.well.com/user/jax/rcfb/forth.html#jeForth

Starting with version 1.00, jeForth can be found at the following site:
http://www.amsystech.com/mlosh/

The jeForth applet and source code are copyrighted freeware released under a license for non-commercial users such as educational institutions and students, non-profit organizations such as the Forth Interest Group and government agencies, and individual users. If you fall into such a category, you may copy and use the jeForth applet on any web server you own or rent hosting services from. You may also copy, use, or modify the jeForth source code for any non-commercial purpose as long as you retain the copyright information and a description of your modifications. The jeForth applet and source code are offered without any warranty and without any specific promise of support. Potential commercial users should contact me for availability and pricing of a future Java-based Forth. Please see the web site for the full terms

(Continues on page 16)

Forth in Control:

Temperature Monitoring

Temperature is one parameter of our environment which has an affect on all living things. We turn up the furnace when we are cold, and switch on the air conditioner in the heat of the summer. Farmers with fruit orchards and cranberry fields watch out for early frost conditions which can decrease their crop yield. Factories closely monitor their process control to ensure the quality of their product is consistent. Machines even perform differently through a range of temperatures. The ignition timing and fuel delivery of your car changes from a cold start to a warmed up engine. Many opportunities arise where there is a need to measure temperature accurately and then perform certain tasks accordingly.

In this article, we will cover how to interface a digital thermometer sensor chip to your computer's parallel port. The device we are using is Dallas Semiconductor's DS1620, which contains the sensor itself and a three-wire serial interface inside an eight-pin DIP package. The sensor measures temperatures from -55 degrees to +125 degrees Celsius in .5 degree increments. This works out to -67 degrees to +257 degrees Fahrenheit in .9 degree increments. Temperature is read from the synchronous serial interface as a nine-bit value. From 0 to 70 degrees Celsius, thermometer error is ±.5 degrees increasing to ±2 degrees at the temperature limit extremes. The sensor is factory set and requires no calibration. The chip has two modes of operation:

1. Three-wire thermometer mode, which communicates ambient temperature data to your computer.
2. Standalone thermostat mode, which needs no computer interface.

Upper and lower temperature values are programmed into the chip's TH and TL nonvolatile EEPROM register memory. The chip has three pins dedicated to alarm outputs, which are active in both modes. T-HIGH goes high when the temperature is greater than the value stored in the TH register. T-LOW goes high when the temperature is less than the value in the TL register. The T-COM pin goes high when the temperature is greater than TH and stays high until the temperature falls below that of TL. In this way, any amount of *hysteresis* can be obtained.

When designing a threshold detector circuit, it is good practice to incorporate hysteresis into the trigger point. If you don't, as the temperature slowly approaches the single trigger point, and passes through, it will flutter just below and just above, causing output "chatter."

To solve this chatter problem, we make two trigger thresholds, T-HIGH which will turn *on* the output, and T-LOW which will turn *off* the output. The difference between these two thresholds is the hysteresis. As an example, when you set your furnace control to 70 degrees, the furnace will run until it hits 72 degrees and will stay off until the temperature drops to 68 degrees. This keeps your furnace from cycling off and on around 70 degrees.

We will use Forth (F-PC) to control the interface hardware through the parallel printer port. [Listing begins on page 17.] All commands to and from the DS1620 sensor will be buffered by the 74LS38 chip. This interface will have the capability to program all EEPROM memory locations in the DS1620 chip for custom configurations and standalone operation.

Pin #1 of the chip is the bi-directional data line (DQ). Data is read to and from the chip via this pin. Data over the interface is communicated LSB first.

Pin #2 is the clock input to which the data is synchronized. The clock transitions are used to determine when to read or send data. A clock cycle is a sequence of a falling edge followed by a rising edge. The data line goes to a high impedance state while the clock is high.

Pin #3 is the Reset input line. All data transfers are initiated by driving the RST line high. Driving the line low terminates communications by forcing the data line into a high impedance mode.

Power (5 volts) is applied to pin #8 (Vcc) and Gnd to pin #4.

Pin #5 is the High/Low combination trigger output. It goes high when the temperature exceeds TH, and resets to low when temperature falls below TL.

Pin #6 is the Low temperature trigger output. It goes high when the temperature falls below TL.

Pin #7 is the High temperature trigger output. It goes high when the temperature exceeds TH.

Data sheets for the DS1620 can be downloaded from Dallas Semiconductor's web site (http://www.dalsemi.com)

Three-Wire Thermometer Mode

Build up the circuit as per schematic [page 15]. A nine-volt battery or an AC power adapter can be used for the power supply. Use an IC socket to hold the DS1620 chip, so it can be easily removed after programming. Because this is a synchronous serial link, we must remember to keep the cable between the computer and circuit board as short as possible. We are using the edges of the clock line as a sync signal for transmitting and receiving data. Extra long lines pick up noise, introduce crosstalk between wires, and increase line capacitance, which can cause data to be corrupted.

Plug the DB25 connector into your parallel printer port and power up the board. Run F-PC and, at the "ok" prompt, type `FLOAD DS1620.SEQ`. If no errors are encountered, type `SHOW.TEMP`. A simple display will appear which will continuously show ambient temperature. To verify correct operation

Ken Merk • Langley BC, Canada
krem@vancouver.net

Ken Merk works for Canadian Pacific Railway. His "Forth In Control" appears periodically in *Forth Dimensions*; see issues XVII.2, XVIII.4, and XIX.4 for other installments in the series.

place your finger on the DS1620 chip, which should cause the temperature reading to slowly increase. (A hair dryer will cause a faster response.) Place a cold object on the chip to see the temperature reading drop. All temperature readings should change in .5 degree increments. Press any key to exit.

The DS1620 is continuously performing temperature conversions and storing the results in a holding register. We read the contents of the register to update our temperature display. To access the register we send a "Read Temperature" command byte (AA Hex) over the serial link. The next sixteen clock cycles will output the contents of this register. The temperature is coded in a two-byte format. The most-significant byte holds the sign bit, and the least-significant byte holds the value of the actual temperature. If the sign bit is high, the temperature is negative and the actual temperature value is in two's complement form. If the sign bit is low, the temperature is positive and the actual temperature value is contained in the least-significant byte.

Temperature data can be logged and stored in a file for future reference, or certain tasks can selected depending on temperature values. [Table One]

Table One. DS1620 nine-bit temp/data

Temp	Binary Value		Decimal	Hex
+125	00000000	11111010	00 250	00 FA
+25	00000000	00110010	00 50	00 32
+0.5	00000000	00000001	00 01	00 01
0	00000000	00000000	00 00	00 00
-0.5	00000001	11111111	01 255	01 FF
-25	00000001	11001110	01 206	01 CE
-55	00000001	10010010	01 146	01 92
	Sign byte*	Temp byte		

* Only the low bit of the sign byte is used, the other seven bits are zeros.

Standalone Mode

In the standalone mode, the DS1620 continuously does temperature conversions and compares them to the pre-programmed threshold values. In this configuration, the chip can monitor temperatures on its own and drive control relays or alarm circuits directly. A computer could poll these outputs, if all you need is a temperature limit detector. Even if your computer misses an alarm output, the DS1620 has a set of Temeratures High/Low flags which remember if a temperature threshold has ever been exceeded. The flags will remain high until reset by writing a zero into this location or by removing power from the device. This feature provides a method of determining whether the DS1620 has ever been subjected to temperatures above threshold limits. These two flags are mapped as bits 5 and 6 in the configuration register. [Table Two]

Table Two. Configuration register map

Bit	Name	Function
0	1 Shot	0 = continuous conversions 1 = 1-shot conversion
1	CPU	0 = standalone mode 1 = three-wire mode
2	X	Don't care
3	X	Don't care
4	X	Don't care
5	TL Flag	0 = Temp > TL 1 = Temp ≤ TL
6	TH Flag	0 = Temp < TH 1 = Temp ≥ TH
7	Done	0 = conversion in progress 1 = conversion done

To configure the chip for standalone operation, we must first program the T-low and T-high values into the chip, and then enable the standalone mode.

To program the Threshold registers, we need a sign byte and a temperature value byte on the stack, then type `WRITE.TH` or `WRITE.TL`. As an example, if we wanted to program T-high with +25.5 degrees, the sign byte would be 00 and the temp value byte would be (25.5 * 2) = 51 (33 Hex).

```
00 51 WRITE.TH
```

If we wanted to program T-low with -25.0 degrees, the sign byte would be 01, and the temp value byte would be (25 * 2) = 50, then invert and add 1 ---> 206 (CE Hex).

```
01 206 WRITE.TL
```

To verify your values, type `SHOW.PARAM` to display the threshold temperatures:

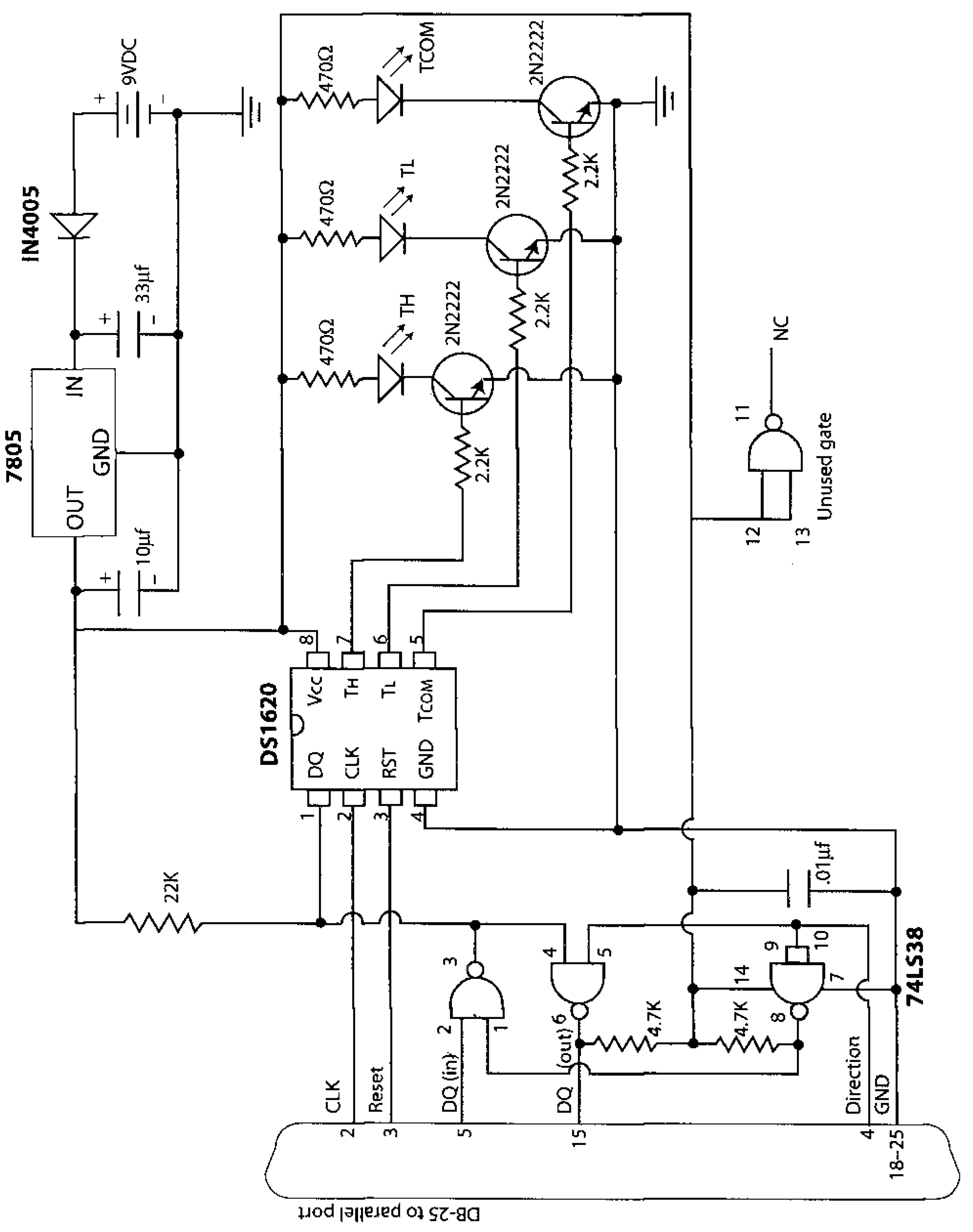
```
T-HIGH --->  25.5
T-LOW  ---> -25.0
```

Program T-low and T-high a few degrees above and below room temperature. Force the temperature trigger points by heating and cooling the chip, and watch the output LEDs change accordingly. Type `SHOW.PARAM` to see the temperature limit flags. They both (bit 5 and 6) should be high, indicating that the values were exceeded.

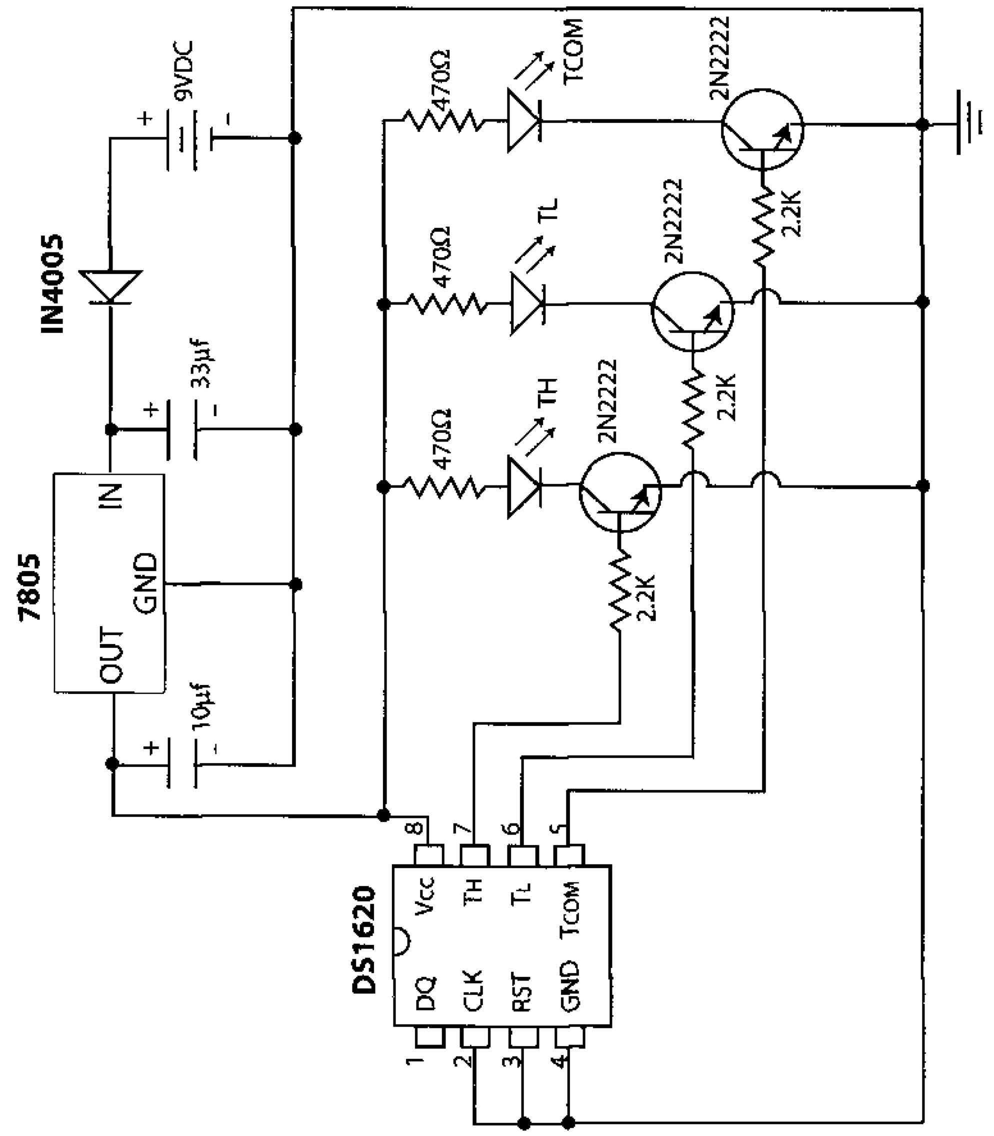
```
CONFIG ---> 01100000
```

To enable the standalone mode, type `STAND.ALONE`. The chip is now ready to be embedded into your custom temperature control application using the standalone circuit as per schematic [page 16].

Three-Wire Mode



Standalone Mode



(eForth for Java, from page 12)

of the license.

Please watch the Usenet newsgroup comp.lang.forth for future announcements on jeForth and any FIG-sponsored projects. I am available via e-mail at mlosh@tir.com for questions or comments.

Future Directions
(but not a conclusion)

The current version of jeForth demonstrates that Java provides a good platform for demonstrating Forth on the Internet. I hope that the Forth community will recognize the opportunity to promote Forth through Java. Furthermore, I hope the Forth Interest Group will organize a project to develop a compelling web site with a great tutorial for the jeForth applet. I am willing to guide such a project, but I would like the ideas and the expertise of others. Please contribute your time and talent to this endeavor!

Michael A. Losh has been a Forth enthusiast since stumbling upon Leo Brodie's books in a physics laboratory in 1990. Currently he is a Microsoft Certified Trainer for Windows programming with the Win32 API, Visual C++, and MFC, and is the director of software consulting at American Systems Technology, Inc. (www.amsystech.com), a Microsoft Solutions Provider and Microsoft Authorized Technical Education Center near Detroit, Michigan.

Listing One. Forth in Control: Temperature Monitoring

```
\ DS1620.SEQ                                              Ken Merk May/98
\ F-PC

\               Forth Code to Interface DS1620 Thermometer
\               ******************************************

   DECIMAL

     $0040 $0008 @L                    \ Look for active LPT1 port
              0= #IF                   \ If no port found then abort

              CLS
              23 8 AT .( Parallel printer port not found.)
              CLOSE QUIT

                 #ENDIF

     $0040 $0008 @L CONSTANT #PORT     \ Find port addr for printer card
                                       \ assign to constant #PORT

  0   VALUE  TEMP                   \ Temp data storage
  0   VALUE  MASK                   \ Mask value
  0   VALUE  NEG                    \ Neg flag
  1   CONSTANT CLK                  \ assign weighting
  2   CONSTANT RST                  \ to CLK and RST
  4   CONSTANT DIRECTION            \ Direction bit
  8   CONSTANT DQ                   \ Data I/O

  code bset  ( b  #port -- )      \ will SET each bit in #port that matches

  pop dx                          \ every high bit in byte b.
  pop bx
  in  ax, dx
  or al, bx
  out dx, al
  next
  end-code


  code breset  ( b  #port -- )    \ will RESET each bit in #port that matches
  pop dx                          \ every high bit in byte b.
  pop bx
  not bx
  in  ax, dx
  and al, bx
  out dx, al
  next
  end-code


  : HIGH       ( b  -- )      #PORT bset    ;   \ turn ON output
  : LOW        ( b  -- )      #PORT breset  ;   \ turn OFF output
  : WRITE      ( b  -- )      #PORT PC!     ;   \ write byte to data port

  : DQ.HIGH   ( -- )     DQ HIGH           ;   \ write 1 -- >  DQ bit
  : DQ.LOW    ( -- )     DQ LOW            ;   \ write 0 -- >  DQ bit

  : DQ.HIGH?  ( -- f)
              08 #PORT 1+ PC@ AND 0<>  ;       \ read DQ bit --high?
```

```
: PORT.INIT  ( -- )  05 WRITE    ;          \ CLK=1 RST=0 DIR=1

: DELAY      ( -- )  1 MS  ;                \ create 1 MS delay

\ To check the accuracy of the MS time delay which can vary with computer
\ speed, type TIMER 2000 MS and check that the time delay is 2 seconds.
\ Adjust by changing the variable FUDGE accordingly.

: PULSE.CLK          ( -- )
                     CLK LOW                    \ CLK=0
                     DELAY                      \ 1 MS delay
                     CLK HIGH                   \ CLK=1
                     DELAY  ;                   \ 1 MS delay


: INVERT.BYTE  ( b1 -- b2)    $FF XOR ;

: INVERT.WORD  ( u1 -- u2)    $FFFF XOR ;


: WRITE.BYTE          ( b -- )
                      1 =: MASK
                      INVERT.BYTE
                      DIRECTION LOW                \ write direction
                      8 0  DO dup MASK AND 0=      \ send eight bits
                                IF    DQ.LOW  PULSE.CLK
                                ELSE  DQ.HIGH  PULSE.CLK
                                THEN  MASK  2*  !>  MASK    \ shift left
                           LOOP  drop ;


: READ.BYTE           ( -- b)
                      0 =: TEMP
                      DIRECTION HIGH              \ read direction
                      8 0 DO                      \ read eight bits
                            CLK LOW  DELAY
                            DQ.HIGH?
                            CLK HIGH DELAY
                            TEMP 2/ !> TEMP                \ shift right
                            IF TEMP 128 OR !> TEMP THEN
                          LOOP
                            TEMP
                            INVERT.BYTE  ;


: READ.WORD       ( -- u)
                  0 =: TEMP
                  DIRECTION HIGH              \ read direction
                  16 0 DO                     \ read sixteen bits
                         CLK LOW  DELAY
                         DQ.HIGH?
                         CLK HIGH DELAY
                         TEMP U2/ !> TEMP               \ shift right
                         IF TEMP 32768 OR !> TEMP THEN
                       LOOP
                         TEMP
                         INVERT.WORD ;


: WRITE.TH        ( b1 b2 -- )        \ write T-High reg
                  RST HIGH
```

FORTH INTEREST GROUP MAIL ORDER FORM

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

FORTH DIMENSIONS BACK VOLUMES

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

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

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

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

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

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

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

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

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

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

Fractal landscapes, stack error checking, perpetual date routines, headless compiler, execution security, ANS-Forth meeting, computer-aided instruction, local variables, transcendental functions, education, relocatable Forth for 68000.

Volume 10 *Forth Dimensions* (1988–89) 110 - $35

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

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

Local variables, graphic filling algorithms, 80286 extended memory, expert systems, quaternion rotation calculation, multiprocessor Forth, double-entry bookkeeping, binary table search, phase-angle differential analyzer, sort contest.

Volume 12 *Forth Dimensions* (1990–91) 112 - $35

Floored division, stack variables, embedded control, Atari Forth, optimizing compiler, dynamic memory allocation, smart RAM, extended-precision math, interrupt handling, neural nets, Soviet Forth, arrays, metacompilation.

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

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

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

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

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

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

FORML CONFERENCE PROCEEDINGS

FORML (Forth Modification Laboratory) is an educational forum for sharing and discussing new or unproven proposals intended to benefit Forth, and is for discussion of technical aspects of applications in Forth. Proceedings are a compilation of the papers and abstracts presented at the annual conference. FORML is part of the Forth Interest Group.

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

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

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

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

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

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

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

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

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

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

BOOKS ABOUT FORTH

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

This set of 11 lessons is designed to make it easy for you to learn Forth. The material was developed over several years of teaching Forth as part of a senior/graduate course in the design of embedded software computer systems at Oakland University in Rochester, Michigan. *156 pp.* (w/disk)

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

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

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

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

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

EXCITING NEW TITLE!

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

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

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

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

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

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

LIMITED TIME!

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

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

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

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

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

WRITING FCODE PROGRAMS **252 – $52**

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

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

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

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

Fast service by fax: 408.373.2845

LEVELS OF MEMBERSHIP

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

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

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

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

DISK LIBRARY
Contributions from the Forth Community

The "Contributions from the Forth Community" disk library contains author-submitted donations, generally including source, for a variety of computers & disk formats. Each file is designated by the author as public domain, shareware, or use with some restrictions. This library does not contain "For Sale" applications. *To submit your own contributions, send them to the FIG Publications Committee.*

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Version-Replacement Policy

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

MORE ON FORTH ENGINES

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

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

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

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

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

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

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

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

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

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

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

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

MISCELLANEOUS

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

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

Last 5

DR. DOBB'S JOURNAL back issues

Annual Forth issues, including code for Forth applications.

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

FORTH INTEREST GROUP

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

For credit card orders or customer service:
Phone Orders *408.37.FORTH*
weekdays *408.373.6784*
9.00 – 1.30 PST *408.373.2845 (fax)*

Name ______________________
Company ______________________
Street ______________________ voice ______________
City ______________________ fax ______________
State/Prov.______________ Zip________ e-mail ______________
Nation

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

PRICES MAY CHANGE WITHOUT NOTICE

Item	Title	Quantity	Unit Price	Total

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

Card Number ______________________ exp. date ________

Signature ______________________

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

✖ MEMBERSHIP IN THE FORTH INTEREST GROUP

The Forth Interest Group (FIG) is a worldwide, non-profit, member-supported organization with over 1,000 members and 10 chapters. Your membership includes a subscription to the bi-monthly magazine *Forth Dimensions*. FIG also offers its members an on-line data base, a large selection of Forth literature and other services. Cost is $45 per year for U.S.A; all other countries $60 per year. This fee includes $39 for *Forth Dimensions*. No sales tax, handling fee, or discount on membership.

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

PAYMENT MUST ACCOMPANY ALL ORDERS

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

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

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

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

Fast service by fax: 408.373.2845

```
                $01 WRITE.BYTE      \ send "Write TH" command byte $01
                WRITE.BYTE          \ write temp value - b2
                WRITE.BYTE          \ write sign byte -  b1
                RST LOW
                10 MS     ;         \ wait for eeprom write cycle


: WRITE.TL      ( b1 b2 -- )        \ write T-Low reg
                RST HIGH
                $02 WRITE.BYTE      \ send "Write TL" command byte $02
                WRITE.BYTE          \ write temp value - b2
                WRITE.BYTE          \ write sign byte -  b1
                RST LOW
                10 MS     ;         \ wait for eeprom write cycle


: POS.FORMAT  ( u -- )   space 0 <# # ascii . hold #S #> type 2 spaces  ;

: NEG.FORMAT  ( u -- )   0 <# # ascii . hold #S ascii - hold #> type
                         2 spaces  ;

: .BIN        ( b -- )   base @ >r binary 0 <# # # # # # # # # #> type
                         space r> base !  ;


: SHOW.TH       ( -- )             \ Display T-High value in degrees
                RST HIGH
                $A1 WRITE.BYTE     \ send "Read TH" command byte $A1
                READ.WORD DUP
                RST LOW
                10 MS
                256 AND 0=
                IF  OFF> NEG 10 * U2/
                ELSE ON> NEG $FEFF AND INVERT.BYTE 1+ 10 * U2/
                THEN ."  ---> "  NEG
                        IF    NEG.FORMAT      \ Display Temp
                        ELSE  POS.FORMAT
                        THEN cr     ;


: SHOW.TL       ( -- )            \ Display T-Low value in degrees
                RST HIGH
                $A2 WRITE.BYTE   \ send "Read TL" command byte $A2
                READ.WORD DUP
                RST LOW
                10 MS
                256 AND 0=
                IF  OFF> NEG 10 * U2/
                ELSE ON> NEG $FEFF AND INVERT.BYTE 1+ 10 * U2/
                THEN ."  ---> "  NEG
                        IF    NEG.FORMAT      \ display Temp
                        ELSE  POS.FORMAT
                        THEN cr    ;


: START.CONVERSION  ( -- )
                    RST HIGH
                    $EE WRITE.BYTE     \ send "Start conversion"
                    RST LOW  ;         \ command byte $EE
```

```
: STOP.CONVERSION   ( -- )
                    RST HIGH
                    $22 WRITE.BYTE    \ send "Stop conversion"
                    RST LOW  ;        \ command byte $22


: WRITE.CONFIG      ( b -- )          \ write byte to config reg
                    RST HIGH
                    $0C WRITE.BYTE    \ send "Write config" command
                    WRITE.BYTE        \ byte $0C
                    RST LOW
                    10 MS ;


: SHOW.CONFIG       ( -- )            \ Display config reg in binary
                    RST HIGH
                    $AC WRITE.BYTE    \ send "Read config" command
                    READ.BYTE         \ byte $AC
                    ."  ---> "  .BIN CR
                    RST LOW
                    10 MS ;


: SHOW.PARAM        ( -- )                   \ Display all chip parameters
                    cr cr
                    ." T-HIGH " SHOW.TH      \ display T-High in degrees
                    ." T-LOW  " SHOW.TL      \ display T-Low in degrees
                    ." CONFIG " SHOW.CONFIG  \ display Config reg in binary
                    cr  ;

: STAND.ALONE     ( -- )
                  00 WRITE.CONFIG  ;    \ configure for stand alone mode


: READ.TEMP       ( -- b1 b2)         \ read temp conversion reg
                  RST HIGH
                  $AA WRITE.BYTE      \ send "Read Temp" command byte $AA
                  READ.WORD DUP
                  RST LOW
                  256 AND 0=
                  IF  OFF> NEG 10 * U2/
                  ELSE ON> NEG $FEFF AND INVERT.BYTE 1+ 10 * U2/
                  THEN   ;


: SHOW.TEMP       ( -- )                \ display temperature in degrees
                  PORT.INIT
                  $02 WRITE.CONFIG      \ set up config reg for 3 wire
                  START.CONVERSION
                  1 seconds             \ wait for complete conversion
                  CLS cursor-off
                      BEGIN
                            20 12 at ." TEMP ---> " READ.TEMP
                            NEG  IF   NEG.FORMAT     \ display temp
                                 ELSE POS.FORMAT
                            38 12 at ." Degrees C "
                                 THEN key?           \ hit any key to quit
                      UNTIL  cr  cursor-on
                  STOP.CONVERSION ;
```

LOAD" Module"

This article outlines a method for organising program sections (called *modules*) utilising the top line (line 0) on each screen—the line usually left for "notes" in most screen editors. The idea was primarily developed to organise the loading of programs, but can be used to implement, for instance, a simple help engine.

The program listing is for LMI's UR/FORTH (Forth-83).

This idea had its beginnings in the use of the first line of a screen (line 0) for notes. It seems almost all Forth screen editors use this convention—I have notes going all the way back to 1979 (in Forth-79!) that use this idea.

For some time before I wrote this program, it seemed to me that this first line could be used for much more than notes—it could virtually be used as a record "header," with the rest of the screen as the "data," which led to the idea that a program could scan through screens, using the data on line 0 for structural information.

This thinking was proceeding along with other thoughts about program organisation (and version control and so on), especially the desire to divide programs up into chunks—not necessarily different *files*, just different *sections*. The main problem in using standard loading screens like either:

```
  1 LOAD
 10 LOAD
 20 LOAD etc. (when using -->)
```
or:
```
 1  9 THRU
10 16 THRU
20 27 THRU etc. (when not using -->)
```

is that, when editing screens, especially when inserting and deleting screens, almost *all* the numbers in the above statements have to be changed. We were looking for a system which did not need this editing. If possible, the ideal aim was to do away with screen numbers altogether (at least when defining a loading screen!).

Finally, these ideas came together in what we at Jarrah Computers call *modules*. Each section of a program, called a module, has the name of the module on the top line (line 0) of the screen. Modules consist of a *consecutive* series of screens, with the same identifying string on line 0 of each screen.

The main word in this program is `FINDmodule` which accepts a string as its argument, and then scans the file looking for the first screen with the given string on the first line. If no screen is found, a false flag is left.

If a screen with a matching string is found, the screen number is saved as the start screen, and a second search is undertaken, examining subsequent screens until the match string is *not* found. In this case of a found string, `FINDmodule` leaves the starting screen, the ending screen+1, and a true flag on top. We decided to make the second argument end screen+1 so that the arguments could be fed directly into a `DO LOOP` (after a `SWAP`!).

To handle the "given string," and to make the process nestable, we implement a string buffer which can hold up to ten (*nests*, in fact) strings. Effectively, the string buffer, and the variable `$NEST` make up a string "stack." Utility word `>$BUF` moves a string to the string buffer ("pushing" it onto the string stack), and `.$BUF` displays the (top) of the string buffer. The word `$[` runs `>$BUF` and then nests a level, and the word `]$` unnests. So, the syntax:

```
 " ModuleName" $[ ModuleFUNCTIONs ]$
```

will push the string `ModuleName` onto the string stack, and the `ModuleFUNCTIONs` in the `$[ ]$` brackets will be run using `ModuleName` as their argument. After `]$` the string is popped, leaving the stack in the same state as at entry.

After defining `FINDmodule`, we can easily write a `FIND"` word which is used

```
FIND" aModule" or FIND" bModule"
```

etc., primarily for interactive use.

Having developed `FINDmodule`, all that remains is to put it to use in applications!

The first real use to which we put `FINDmodule` is to `LOAD` screens, as in `LOADmodule`. If the module is found, all we need to say is `1- THRU`. Of course, if it is not found, we simply provide a message. Again, from `LOADmodule`, it is easy to define `LOAD"` (as in `LOAD" ThisOrThat"`). With these tools, we can write a set of loading commands like:

```
LOAD" StartModule"
LOAD" MainFunctionals"
LOAD" DisplayFunctionals"
LOAD" MainProcesses"
LOAD" OuterLoop"
```

to load an entire program—without specifying screen numbers!

Of course, the next thing that happens is that these commands themselves occupy a set of screens, which can be defined as a module! At Jarrah Computers, we call this the *program* module, and it has the format, for example, `" ThisProgram"` or `" ThatProgram"` and is usually the first module defined in a screen file. To load the entire program, all we have to say is

```
LOAD" ThisProgram"
```
(or, alternatively, `LOAD" ThatProgram"`!)

Another simple (but very handy!) function is to define a module `" RELOAD"`, a screen which usually contains only:

```
FORGET FirstWordInTheProgram
 LOAD" TheProgram"
```

Dave Edwards, Perth, WA, Australia
jarrah@inf.net.au

Dave Edwards is a qualified electronic engineer. In 1984, he formed Jarrah Computers, specialising in custom microcontrollers—e.g., the HC05, 65F11/12, HC11 and, of course, Forth chips from the Novix on.

and we can then LOAD" RELOAD".

It is quite easy to extend this system to load modules from other files, all we require is to buffer the filenames (again, nestably), functions which we do not need to cover here. We called this word USE_LOAD", used in the form:

```
" Filename.abc" USE_LOAD" Module"
```

Another handy function we call BROWSEing (included in the listing) has been used to implement a simple help system. All we need to do is FIND the module and, if it is found:

```
START with the StartScreen (returned by a
successful FINDmodule)

BEGIN LIST the current screen, wait for key...
      If pgup: Decrement the current screen
               (bounded by StartScreen)
      If pgdn: Increment the current screen
               (bounded by EndScreen)
      If Esc : exit,  Else continue
```

In this example, we use a simple LIST facility to display the screen, but this could be any desired display (in its own "window," etc). Finally, to implement the help facility, all that is needed is to put phrases like:

```
BROWSE" ThisSectionHelp"
```
or
```
BROWSE" ThatSectionHelp"
```

etc., wherever required in the code. Note that it is very easy to implement the " PAGE UP previous" and " PAGE DOWN next" prompts (which disappear when they are no longer relevant), as we have made copies of the START and END screens. The main advantage of using this system is that the help file can be edited at any time and the program will display all the new screens without need to modify the program (again, provided that the module names have been included on line 0 of each new screen).

Dave Edwards is a qualified electronic engineer who formed Jarrah Computers, an embedded systems development company using Forth, in 1984. His company has specialised in design of custom microcontrollers ranging from the HC05 single-chip family, through the Rockwell 65F11/12, the 68HC11 and, of course, has investigated the range of Forth chips ever since the Novix appeared. Dave presented a paper on applications of the Novix to the Australian Forth Symposium and has previously contributed articles to *Forth Dimensions*.

Dave's other interests include music—both performance and composition. In 1993, he wrote an opera ("Giles - Is That You?") and is currently working on a second opera ("Giles at Fort Meuller"). The MIDI music system used for the two productions was programmed by Dave in Forth especially for the productions, and he still does some session work as a keyboard player around Perth.

Listing One

```
( Code Following)

\ Modules                                       09:57 22.06.98

    Code for LMI's UR/FORTH (F83) for modules of code.

    FIND"   with $Buffer Nesting.

    LOAD"   with No File Nesting.

    BROWSE" for implementing simple Help.

    Version ONLY for ForthDimensions Article.

    Copyright (C) Jarrah Computers 1993-1998.


\ LOAD"                                         09:55 22.06.98

64 CONSTANT CH/LN                 ( Characters/Line )
24 CONSTANT Mod"Size              (ModuleStringSize )
10 CONSTANT NESTS      VARIABLE $NEST $NEST OFF   ( $NestLevel)

CREATE      $BUFS NESTS Mod"Size  * ALLOT (Buffer For Strings)

:  $BUF ( -- A) $BUFS $NEST @ 0 MAX NESTS MIN Mod"Size * + ;
: >$BUF ( A --) $BUF Mod"Size + DUP OFF OVER C@ 1+ CMOVE ;

: $[   >$BUF $NEST @ NESTS 1- < IF  1 $NEST +! THEN ;
: ]$        $NEST @         0> IF -1 $NEST +! THEN ;
: 'SOURCE   [ ' SOURCE >BODY ] LITERAL ; (For Source resetting)
 -->
\ LOAD"                                         09:53 22.06.98

: $onLine0 ( Scr# -- f) \ True if $BUF String on Line0 of Scr#
            BLOCK 2+                ( Point to word After " \_")
            CH/LN 'SOURCE 2! 0 >IN !               ( NewSource )
            $BUF COUNT BL WORD COUNT STRCMP 0= ;   ( $Match? )

: FINDmodule ( -- F | n1 n2 T ) \ n1 StartScr, n2 EndScr+1
            >IN @ >R BLK @ >R SOURCE >R >R        (Save Source)
            FALSE ( Seed the stack)
 ?SCREENS 0 ?DO I $onLine0 IF DROP I TRUE LEAVE THEN LOOP
         IF ?SCREENS 2DUP SWAP ( Now search for EndOfModule)
            ?DO I $onLine0 NOT IF DROP I LEAVE THEN LOOP TRUE
       ELSE FALSE
       THEN R> R> 'SOURCE 2! R> DUP BLK ! BLOCK DROP R> >IN ! ;
-->
\ LOAD"                                         09:58 22.06.98

: .$BUF ( --) \ For displaying Modules as we are loading
            OUT @ 80 $BUF C@ 1+ - > IF CR THEN
            $BUF COUNT TYPE SPACE ;

: LOADmodule ( -- ) \ Loads Screens with $=$BUF
            .$BUF FINDmodule IF 1- THRU
                         ELSE ABORT" .. not found " THEN ;

: FIND"     ASCII " FEED $[  FINDmodule ] $ ;     IMMEDIATE
: LOAD"     ASCII " FEED $[  LOADmodule ] $ ;     IMMEDIATE

( and that's the end of "-->"!)



\ Browse - for implementing Help                09:58 22.06.98
DECIMAL
VARIABLE MODSTART         ( Start Screen of module)
VARIABLE MODEND           ( End   Screen of module)

: DISP-SCR  DUP LIST                          ( n -- n )
            CR ." ESC to exit    "
            DUP MODSTART  @
   = NOT IF ." PAGE UP previous    "
        ELSE ."                    "
        THEN DUP MODEND @
   = NOT IF ." PAGE DOWN next "
        ELSE ."               "
        THEN ;


\ Browse - for implementing Help                09:58 22.06.98

: BLIP      200 15 BEEP ;
: WITHIN    1+ OVER - >R - R> U< ;

: HELP+     DUP MODEND   @ < IF 1+ ELSE BLIP THEN ;  ( n -- n')
: HELP-     DUP MODSTART @ > IF 1- ELSE BLIP THEN ;  ( n -- n')
```

(Code continues on page 26.)

Local Macros

Simple macros can be implemented in Standard Forth with string literals and **EVALUATE**.

```
: :GO S" ANEW NONCE  : (GO) " EVALUATE ;
IMMEDIATE
: GO  S" (GO) NONCE "  EVALUATE ;
IMMEDIATE
```

This means that code **:GO** will be resolved by evaluating **ANEW NONCE : (GO)**. That starts the definition of **(GO)**. When the definition has been completed with **;**, then **GO** will execute **(GO)** and automatically forget it along with **NONCE**.

So macros are shorthand. We have shorthand for creating the shorthand.

```
( Simple Macro -- No parameters. )
: MACRO                    ( "name <char> ccc<char>" -- )
    :   CHAR PARSE POSTPONE SLITERAL  POSTPONE EVALUATE
    POSTPONE ; IMMEDIATE
;
```

The two macros above can be written:

```
MACRO :GO   " ANEW NONCE  : (GO) "
MACRO GO    " (GO) NONCE "
```

Forth macros can be used when interpreting or compiling, and are known globally in a search order.

C macros are compile only. An application can have many macros, which disappear after the compilation. This is convenient for factoring the application for the time now only. We can define macros, use them, and lose them.

Global macros, as defined above, are put into the Forth dictionary. When the Forth interpreter recognizes them, it executes the word and evaluates the associated text. Macros are immediate and so are not compiled. The evaluated text may be.

Local macros are not put into the Forth dictionary. Instead, a common area is used and re-used as files are compiled. The size of the area, **My-Macros-Size**, will depend on your use, and you can increase or decrease it.

The macros are stored as strings—name and what's to be evaluated—in the familiar last-in, first-found sequence. The word **my** followed by a name looks the name up and evaluates the associated string when the name is found. Within local macros, **my** must also be used to resolve other local macros.

You use **CLEAR-MY-MACROS** to empty the list of macros before putting your present ones in.

Local macros are defined similarly to global macros:

```
my MACRO name    " what's to be evaluated  "
```

Any non-blank character may be used instead of ". The delimiter should not occur in the text. I generally use | when " is in the text.

Local macros are not Forth definitions. They do not take any dictionary space. Define them, use them, and throw them away.

In the Stretching Forth article "What's a Character?" local macros moderate what would have been an excessive number of definitions.

The source code includes definitions of **PLACE**, **BUFFER:**, **CHAR-DO**, **CHAR-LOOP**, and **Uppercase-Pad**.

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

Wil Baden is a professional programmer with an interest in Forth. For a copy of the source for this article, send e-mail requesting "Standard Forth Tool Belt #5: Local Macros."

```
( Local Macros )

( User words:
CLEAR-MY-MACROS
my MACRO _newmacroname_  " What to do "
my _macroname_
)

   2000 CHARS CONSTANT  My-Macros-Size  ( Whatever you need. )
My-Macros-Size BUFFER:  My-Macros

MACRO node@ " @ "    MACRO node! " ! "

( Scan for item in a list, one by one. )
( Called by  `my` to find macro. )
: scan-item                 ( str len head -- item | 0 )
    ROT ROT 2>R                             ( list)( R: str len)
        BEGIN  node@  DUP WHILE
               DUP CELL+ COUNT  2R@  COMPARE 0=
        UNTIL  CELL+ ( item)
        THEN
    2R> 2DROP
;

( Check that there's still enough room in macro space. )
( Called by `MY-MACRO`. )
: my-macros-enough               ( n addr -- same )
    OVER 1+ CHARS  OVER +  My-Macros My-Macros-Size +  U>
         ABORT" My Macros Full. "
;

( Make name upper case for case insensitivity. )
( Called by `MY-MACRO` and `my`. )
: raise-case                     ( str len -- str' len )
    31 MIN  Uppercase-Pad PLACE  Uppercase-Pad COUNT
    \ 2DUP chars-to-upper
    2DUP CHAR-DO  I C@ DUP [CHAR] a - 26 U< BL AND - I C! CHAR-LOOP
;

( Find where new macro will go, and link to top of list. )
( Called by `MY-MACRO`. )
: My-New-Macro                                    ( -- addr )
    My-Macros node@                                  ( addr)
    DUP 0= IF  DROP  My-Macros CELL+  ( First macro )
         ELSE  CELL+  COUNT CHARS +  COUNT CHARS +  ALIGNED
         THEN
    DUP My-Macros  2DUP node@ SWAP node! node!
    CELL+
;

( Place macro name and replacement in macro list. )
( Called by `my`. )
: MY-MACRO           ( " name <char> string<char>" -- )
```

```
    My-New-Macro >R                                 ( )( R: addr)
        BL WORD COUNT ( str len) raise-case         ( str len)
        R@  my-macros-enough  PLACE                 ( )
    CHAR PARSE  R> COUNT CHARS +  my-macros-enough  PLACE  ( R: )
;

( Use `my` before local macro names and before `MACRO`. )
: my                                ( " name" -- ??? )
     BL WORD COUNT ( str len) raise-case        ( str len)
     2DUP S" MACRO" COMPARE 0=
          IF  2DROP  MY-MACRO  EXIT  THEN
     My-Macros scan-item  DUP 0= ABORT" Not my macro. "
     COUNT CHARS +  COUNT EVALUATE
; IMMEDIATE

( Start a new set of local macros. )
: CLEAR-MY-MACROS ( -- ) 0 My-Macros ! ;
```

Load" Module" code, continued from page 23.

```
: GET-KEYS      KEY DUP         ( -- n ; converts to uppercase)
         0= IF DROP KEY 8 SHIFT
          ELSE DUP ASCII a ASCII z WITHIN IF 223 AND THEN
          THEN BEGIN ?TERMINAL WHILE KEY DROP REPEAT ;
HEX
001B CONSTANT <ESC>   4900 CONSTANT <PGUP> 5100 CONSTANT <PGDN>
DECIMAL
                                                       -->
\ Browse - for implementing Help                 09:58 22.06.98

( " str" --- ; String of Screens to display )
: BROWSE        $[  CR ." Locating: " .$BUF ( For slow disks! )
  FINDmodule IF 1- MODEND ! DUP MODSTART ! ( Set Screen Limits)
( Scr On Stack) BEGIN DISP-SCR GET-KEYS
                 CASE <ESC>  OF         TRUE  ENDOF
                      <PGUP> OF HELP-   FALSE ENDOF
                      <PGDN> OF HELP+   FALSE ENDOF
                       DUP   OF BLIP    FALSE ENDOF ENDCASE
                UNTIL DROP ( Drop remnant screen#)
           ELSE ." Module Not Found .. " KEY DROP
           THEN ] $ ;               ( Denest at End )

: BROWSE"       ASCII " FEED BROWSE ;       IMMEDIATE
```

STRETCHING STANDARD FORTH – #20

What's a Character?

Forth has all the character and string manipulation functions it needs for interpreting and compiling Forth and for running target systems.

When we want to use Forth for more advanced text handling on host systems, we need more.

One of the first inconveniences is naming. I suspect that converting a character to uppercase is a common function in virtually every system, but there is no common name.

I've used implementations that have called it **>UPPER**, **UPC**, **UPCASE**, **UPPERCASE**, **c>C**.

My solution is to adopt the name from the Standard C Library together with the other related functions. Then I can explain by saying "it's the same as the Standard C Library."

isalnum	Alpha-numeric character
isalpha	Upper- or lower-case letter
iscntrl	Control character
isdigit	Decimal digit
isgraph	Not space
islower	Lowercase
isprint	Printing character, including space
ispunct	Neither space nor letter nor digit
isspace	Space, tab, return, linefeed
isupper	Uppercase
isxdigit	Sedecimal digit
tolower	Convert and return lower-case letter.
toupper	Convert and return upper-case letter.

Here is a minimum storage high-level implementation of those.

```
( CTYPE Functions -- Short )

ANEW --CTYPE-- DECIMAL  ( Slow version -- will be overlayed. )
                                      ( char -- flag )
: isalpha   BL OR [CHAR] a - 26 U< ;
: iscntrl   1+  127 AND  34 < ;
: isdigit   [CHAR] 0 -  10 U< ;
: isalnum   DUP isalpha ORIF DUP isdigit THEN NIP ;
: isgraph   [CHAR] ! -  94 U< ;
: islower   [CHAR] a -  26 U< ;
: isprint   BL -  95 U< ;
: isupper   [CHAR] A -  26 U< ;
: ispunct   DUP isgraph ANDIF DUP isalnum NOT THEN NIP ;
: isspace   DUP BL = ORIF DUP 9 - 5 U< THEN NIP ;
: isxdigit  DUP isdigit ORIF DUP BL OR [CHAR] a - 6 U< THEN NIP ;
                                     ( char -- char' )
: toupper   DUP [CHAR] a -  26 U< BL AND - ;
: tolower   DUP [CHAR] A -  26 U< BL AND + ;
```

The most used of those is **toupper**. If you're going to be doing a lot of text massaging, this should be improved.

If your system has a CODE version of this function, you can adopt it.

```
MACRO toupper  " UPCASE "
```

Another approach is to use a translation table. [Continues on next page.]

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

Wil Baden is a professional programmer with an interest in Forth. For a copy of the source for this article, send e-mail requesting "Stretching Forth #20: What's a Character?."

```
ANEW --CTYPE-- DECIMAL  ( Fast version.  Let's keep this one. )

256 CHARS BUFFER: Uppercase-Table

:GO 256 0 DO
    I DUP [CHAR] a - 26 U< BL AND - Uppercase-Table I CHARS + C!
LOOP ; GO

MACRO toupper  " CHARS Uppercase-Table + C@ "
```

This is much faster than the CODE version on the system I'm using, and I have enough space, so the translation table version is the one I have adopted.

(The code gets optimized in the loop cycle and I can't tell how long it takes.)

```
: MIL 1000000 * ;

: NOTHING ; IMMEDIATE

MACRO toupper  " CHARS Uppercase-Table + C@ "

: >UPPER    DUP [CHAR] a -  26 U< BL AND - ;

: WITHIN    OVER - >R - R> U< ;
: BETWEEN   1+ WITHIN ;
: UPPERCASE DUP [CHAR] a [CHAR] z BETWEEN BL AND XOR ;

// :GO COUNTER 1 MIL 0 DO 127 32 DO I | DROP LOOP LOOP TIMER ; GO CR |
    NOTHING  toupper  UPCASE  >UPPER  UPPERCASE
\\

 \ 3833 MS
 \ 3834 MS
 \ 4783 MS
 \ 10300 MS
 \ 18734 MS
```

I also adopted a translation table for **tolower**.

```
256 CHARS BUFFER: Lowercase-Table

:GO 256 0 DO
    I DUP [CHAR] A - 26 U< BL AND + Lowercase-Table I CHARS + C!
LOOP ; GO

MACRO tolower  " CHARS Lowercase-Table + C@ "
```

The string conversion routines are:

```
                                                ( str len -- )
: chars-to-upper  CHAR-DO  I C@ toupper I C!  CHAR-LOOP ;
: chars-to-lower  CHAR-DO  I C@ tolower I C!  CHAR-LOOP ;
```

To accelerate the character-tests, a 256-byte table of bit-codes is used. Macros set and test those codes. 15 macros are defined to do this. These macros have no other purpose outside these definitions and at this time.

These macros are set up as *local macros*. Their definitions will go away and the space for them will be recovered. There's less here than meets the eye.

The only definitions that will remain will be **Char-Code** and the 11 ***issomething*** functions.

See Tool Belt #5, "Local Macros," for code for local variables. That should be loaded first. **//** is iterated interpretation, presented in Tool Belt #3.

```
( CTYPE   Character Type Functions -- Fast )

CREATE Char-Code   ( Table for character codes. )

CLEAR-MY-MACROS

( Character Testing Functions )

my MACRO Control-Char  " 1 "
my MACRO Motion-Char   " 2 "
my MACRO Space-Char    " 4 "
my MACRO Punctuation   " 8 "
my MACRO Digit         " 16 "
my MACRO Hex-Digit     " 32 "
my MACRO Lower-Case    " 64 "
my MACRO Upper-Case    " 128 "

( Nothing has been compiled since `Char-Code`; so the following
( bytes go there.  Each byte has one bit on. )

// my | C, |

     Control-Char Control-Char Control-Char Control-Char
     Control-Char Control-Char Control-Char Control-Char
     Control-Char Motion-Char  Motion-Char  Motion-Char
     Motion-Char  Motion-Char  Control-Char Control-Char

     Control-Char Control-Char Control-Char Control-Char
     Control-Char Control-Char Control-Char Control-Char
     Control-Char Control-Char Control-Char Control-Char
     Control-Char Control-Char Control-Char Control-Char

     Space-Char   Punctuation  Punctuation  Punctuation
     Punctuation  Punctuation  Punctuation  Punctuation
     Punctuation  Punctuation  Punctuation  Punctuation
     Punctuation  Punctuation  Punctuation  Punctuation

     Digit        Digit        Digit        Digit
     Digit        Digit        Digit        Digit
     Digit        Digit        Punctuation  Punctuation
     Punctuation  Punctuation  Punctuation  Punctuation

     Punctuation  Upper-Case   Upper-Case   Upper-Case
     Upper-Case   Upper-Case   Upper-Case   Upper-Case
     Upper-Case   Upper-Case   Upper-Case   Upper-Case
     Upper-Case   Upper-Case   Upper-Case   Upper-Case
```

```
    Upper-Case   Upper-Case   Upper-Case   Upper-Case
    Upper-Case   Upper-Case   Upper-Case   Upper-Case
    Upper-Case   Upper-Case   Upper-Case   Punctuation
    Punctuation  Punctuation  Punctuation  Punctuation

    Punctuation  Lower-Case   Lower-Case   Lower-Case
    Lower-Case   Lower-Case   Lower-Case   Lower-Case
    Lower-Case   Lower-Case   Lower-Case   Lower-Case
    Lower-Case   Lower-Case   Lower-Case   Lower-Case

    Lower-Case   Lower-Case   Lower-Case   Lower-Case
    Lower-Case   Lower-Case   Lower-Case   Lower-Case
    Lower-Case   Lower-Case   Lower-Case   Punctuation
    Punctuation  Punctuation  Punctuation  Control-Char

\\
    128 RESERVE  ( Clear the rest of the table. )

( Include hex-digits in `Char-Code` table. )

// CHAR | CHARS Char-Code + DUP C@ my Hex-Digit OR SWAP C! |
   0 1 2 3 4 5 6 7 8 9 A B C D E F a b c d e f
\\

my MACRO Letter        " my Lower-Case   my Upper-Case   OR "
my MACRO Alphanumeric  " my Letter       my Digit        OR "
my MACRO Graphic       " my Alphanumeric my Punctuation  OR "
my MACRO Printable     " my Graphic      my Space-Char   OR "
my MACRO Whitespace    " my Motion-Char  my Space-Char   OR "
my MACRO Control       " my Motion-Char  my Control-Char OR "

     my MACRO Char-Code  " CHARS Char-Code + C@ "

                                      ( char -- flag )
: isalnum  my Char-Code  my Alphanumeric AND 0<>  ;
: isalpha  my Char-Code  my Letter       AND 0<>  ;
: iscntrl  my Char-Code  my Control      AND 0<>  ;
: isdigit  my Char-Code  my Digit        AND 0<>  ;
: isgraph  my Char-Code  my Graphic      AND 0<>  ;
: islower  my Char-Code  my Lower-Case   AND 0<>  ;
: isprint  my Char-Code  my Printable    AND 0<>  ;
: ispunct  my Char-Code  my Punctuation  AND 0<>  ;
: isspace  my Char-Code  my Whitespace   AND 0<>  ;
: isupper  my Char-Code  my Upper-Case   AND 0<>  ;
: isxdigit my Char-Code  my Hex-Digit    AND 0<>  ;
```

Part II

Adaptive PID

Introduction

Let's continue with our investigation of adaptive PID controllers by looking at how to implement the plant identification part of such controllers. For those of you that have lost track, PID controllers achieve the goal of regulating a system by combining a signal that is *proportional* to the error input, plus an *integral* of the error signal, and the *derivative* of the error signal. Controlling by just using the proportional signal tends to cause oscillations; adding the integral term reduces these. Adding the derivative term makes the system more responsive to signal changes. If we know enough about the system that we are controlling, then, in principle, we can adjust the gains for the separate proportional integral and derivative terms to achieve an optimal (critically damped) controller.

The problem with this is that one is often faced with a system that is not characterized well enough to do this, or the system that is under control has unsteady parameters. When we are faced with this situation, an adaptive controller is a good choice to handle it.

Mathematical recap

First let's take a look at quick summary of the equations.

Our controller is,

$$y(t) = K_p\varepsilon(t) + K_i\int_0^t \varepsilon d\tau + K_d\frac{d\varepsilon(t)}{dt} \qquad (1)$$

where K_p is the proportional gain, K_i is the integral gain, and K_d is the differential gain. The quantity ε is an error signal that is the difference between the commanded input, x and the output of the controlled plant z.

Our controlled plant is defined by the differential equation,

$$F(z) = \alpha\frac{d^2z}{dt^2} + \beta\frac{dz}{dt} + \gamma z \qquad (2)$$

where α, β, and γ are known constants and $F(z)$ represents the imposed external forces on the plant (the input).

The controller and the plant are coupled by,

$$\varepsilon = (x - K_{fb}z)\,K_0 \qquad (3)$$

where K_0 is a known input gain of the controller and K_{fb} is a known feedback gain that is output from the plant.

Note that we have changed variable names slightly, as compared to last time, so that we can consistently hook everything up together. To summarize: x is the input signal, y is the controller output *and* the plant input, and z is the plant output. The controller input is ε as defined in (3).

The optimization

The controller is optimally tuned when the expected mean squared value of system error is minimized (i.e., we are doing another *least-squares* problem). It is *very* easy to get lost here and lose track of what we are doing, so let's be explicit about what we are dealing with. We are combining an input signal, x with the *actual* plant output, z, to create an error signal ε. The error signal is then used as a control input, y, into the actual plant. If we knew the characteristics of the actual plant, α, β, and γ, we could adjust the PID controller gains K_p, K_i, and K_d so that it is critically damped.

In the last installment, we derived the equations necessary to achieve the proper gains given the plant parameters. However, in our current scenario, we do not know the actual plant parameters. So we *estimate* the plant parameters, given y and z. We then use these estimated plant parameters to choose our controller gains.

So we want to minimize the mean square of the difference between the output of the actual plant and the currently estimated plant,

$$J(\alpha,\beta,\gamma) = \sum_{k=1}^{n}(\text{error})^2 = \sum_{k=1}^{n}(Z - z_k)^2 \qquad (4)$$

where Z is the *actual* plant output and z_k is the modeled plant output. Now. since we are going to be creating a digital implementation of the controller, at some point we are going to have to switch from using differential equations to finite difference approximations to them. If we make that switch at this point in our analysis, things will be somewhat simpler so, using second order finite difference approximations, our model plant equation becomes

$$z_t = \left(\frac{4\alpha - 2\gamma H^2}{\beta H + 2\alpha}\right) z_{t-1} + \left(\frac{\beta H - 2\alpha}{\beta H + 2\alpha}\right) z_{t-2} + \frac{2H^2}{\beta H + 2\alpha} y_{t-1} \qquad (5)$$

(where H is the time step size), simplifying the notation again,

$$z_t = a_1 z_{t-1} + a_2 z_{t-2} + b_1 y_{t-1} \qquad (6)$$

or more generically, we can write this as,

$$z_t = \sum_{k=1}^{p} a_k z_{t-k} + \sum_{k=1}^{q} b_k y_{t-k} \qquad (7)$$

Using this form, we can always go back to the original

Skip Carter • Monterey, California
skip@taygeta.com

Skip Carter, a scientific and software consultant, and leader of the Forth Scientific Library project, maintains www.taygeta.com. He is also the President of the Forth Interest Group.

parameters,

$$\begin{aligned} \alpha &= -\frac{H^2}{2b_1}(a_2 - 1) \\ \beta &= \frac{H}{b_1}(a_2 + 1) \\ \gamma &= \frac{1 - a_1 - a_2}{b_1} \end{aligned} \quad (8)$$

The steps we go through next are exactly the same as we did for solving the least-squares straight line problem in *FD* XIX.3. Once again we can use a symbolic mathematics package (I use Mathematica) to help avoid making an error in the derivation of the equations for the next steps, which are straightforward but rather tedious to do.

We need to determine when the derivative of J with respect to the parameters is zero,

$$\begin{aligned} \frac{\partial J}{\partial a_1} &= 0 \\ \frac{\partial J}{\partial a_2} &= 0 \\ \frac{\partial J}{\partial b_1} &= 0 \end{aligned} \quad (9)$$

These expand out to equations (10), (11), and (12).

$$\begin{aligned} a_1 \sum z_{t-1}^2 + a_2 \sum z_{t-1}z_{t-2} + b_1 \sum z_{t-1}y_{t-1} &= \sum Z z_{t-1} \quad (10) \\ a_1 \sum z_{t-1}z_{t-2} + a_2 \sum z_{t-2}^2 + b_1 \sum z_{t-2}y_{t-1} &= \sum Z z_{t-2} \quad (11) \\ a_1 \sum z_{t-1}y_{t-1} + a_2 \sum z_{t-2}y_{t-1} + b_1 \sum y_{t-1}^2 &= \sum Z y_{t-1} \quad (12) \end{aligned}$$

which need to be solved for a_1, a_2, and b_1.

Let's step back and take a look at what we have arrived at. We now have three equations and three unknowns, so unless one of equations (10) through (12) turn out to be redundant (they are not), we can ultimately manipulate them to solve for our unknown terms. Further, we can see that our solutions are going to give us the plant parameters α, β, and γ solely in terms of the histories (because of the sums) of the controller output y, the actual plant output signal Z, and the previous estimated plant output signal values z.

The need to maintain the histories creates something of a problem because the terms in (10) through (12) will have to be reevaluated at each time step, which will impact the performance of the controller in a real-time environment. The equation in its general form (7) is known as an ARMA (Auto-Regressive Moving Average) model. ARMA models are extremely important models for discrete systems and appear in many contexts. Techniques for efficiently solving ARMA models were worked out in the '50s, when you just couldn't throw a couple more MIPS and megabytes at the problem. One of the most suitable methods is called *Plackett's algorithm* (Plackett, 1950), which takes a current estimate of the parameters and combines them with the new data to get the new estimate. It is important to recognize that Plackett's algorithm is *not* an approximation to the solution of (10) through (12), it is mathematically exactly the same—it just *looks* very different.

Deriving Plackett's algorithm without the use of matrix or linear algebra is tedious in the extreme, so I won't derive it here. Using linear algebra and something called the Gauss-Markov theorem (scary sounding words, but it's really just matrix-speak for linear least squares, which we already understand), we arrive at the set of *matrix* equations,

$$\mathbf{P}_t = \mathbf{P}_{t-1} - \frac{\mathbf{P}_{t-1}\mathbf{x}_{t-1}\mathbf{x}_{t-1}^T\mathbf{P}_{t-1}}{1 + \mathbf{x}_{t-1}^T\mathbf{P}_{t-1}\mathbf{x}_{t-1}} \quad (13)$$

$$\theta_t = \theta_{t-1} - \frac{\mathbf{P}_{t-1}\mathbf{x}_{t-1}(\mathbf{x}_{t-1}^T\theta_{t-1} - \mathbf{y}_t)}{1 + \mathbf{x}_{t-1}^T\mathbf{P}_{t-1}\mathbf{x}_{t-1}} \quad (14)$$

The new quantities are:

- x_t is a *vector* containing the inputs and plant outputs—the z and y values in (7)—stacked one above the other.
- P_t is the covariance *matrix* of the estimation error. It quantifies how good the current estimate is. It is calculated from the output statistics of the model. The covariance matrix is where the controller history information went to from the direct formulation.
- q_t is a *vector* containing the current plant parameter—the a and b values in (7)—estimates stacked up.

This is what is known as a *recursive estimator*, it makes a new estimate based upon the current estimates plus the new data.

Equations (13) and (14) give a practical method to estimate the plant parameters, given the past statistics and the new data. In this installment, we will look at how to implement this; next time, we will go to the final step and use this estimate to adapt the controller.

The numerical implementation

For all the messy, complicated math, the implementation of all this is actually pretty straightforward once you have the equations and a Forth version of the linear algebra operations. The matrix inverse (the matrix algebra version of divide) is part of the Forth Scientific Library; the other matrix operations we have to write ourselves.

When writing a *simulation* of an adaptive controller, you have to remember to simulate the plant, too. In an actual application, this part would be replaced with a sub-system that aquires the digitized data from the physical system.

Listing One [not available at press time, the code will be available via FTP and will be printed in our next issue. —Ed.] is a ANS Forth implementation that will demonstrate a PID controller plant identification using a simulated input signal. Just to make it interesting, the (simulated) real plant actually changes a couple of times, so the estimate is forced to change, too. Initially, we have no idea what the model parameters are; we describe this uncertainty quantitatively by initializing the covariance matrix P, to be large values on the diagonal and zero off the diagonal.

The rest of the program directly implements equations (13) and (14). The program is designed to create an output time series that can be captured and used with gnuplot.

Tricks to make it work better

The implementation of an adaptive controller often has extra features I have not described so far, such as not recalculating the gains every time step, but only after an interval of several steps. Once this is done, the accumulated sums are also reset, thus reinitializing the adaption section. Doing this helps tame the controller, particularly in a noisy system where it would eventually try to adapt to the noise. This is just one of many practical issues that make the use of adaptive PID controllers both a science and an art.

There are two other common tricks, both of which are intended to keep the integral term under control. Remember that the purpose of the integral term is to smooth out oscillations, but the fact that it is accumulated over the entire run time of the system can cause problems with responsiveness. The first trick is to limit the size the integral can grow to; this is especially valuable to do when the controller is implemented in scaled-integer or fixed-point arithmetic. The second useful trick is to reset the integral to zero when its sign is opposite of the error, ε; this makes the integral term able to respond more quickly to the changing system, which is why the error changed sign (thanks to Jerry Avins for pointing out this one).

Conclusion, Part II

We have now achieved the ability of being able to identify the plant for a PID controller based upon its response to the incoming data. This is the final background piece we need in order to get to the ultimate: understanding and implementing an adaptive PID controller in Forth. I want to re-emphasize the fact that the *details* of what we have arrived at in this example are very dependent upon the choices we made for the plant model and how it is linked into the controller. Lots of other configurations are possible; the choice depends strongly upon the application. If you try to compare this with other derivations in the literature you will almost certainly see something different. What you *will* see in a comparison is that the methods used are basically the same.

Feedback

Please don't hesitate to contact me through *Forth Dimensions* or via e-mail if you have any comments or suggestions about this or any other Forthware column.

References

Dutton, K., S. Thompson, and B. Barraclough, 1997; *The Art of Control Engineering,* Addison-Wesley, Reading Mass. ISBN 0-201-17545-2

Plackett, R.L., 1950; Some theorems in least squares, *Biometrica,* V. 37, pp. 149–157.

Contact Conference Organiser or Conference Chair for details.

euroFORTH '98

The 14th euroFORTH conference on the FORTH programming language and FORTH processors (Including an Internationalisation Workshop)

September 18–21, 1998

Conference delegates are welcome and encouraged to give papers on subjects related to the conference topics. As usual there will be a "4th" day, which will include an exhibition (DM 100 per stand) and a chance for delegates to review the conferences. As in the previous years delegates from all parts of Europe and other continents are expected.

Internationalisation Workshop

When the International Organisation for Standardisation accepted ANS Forth as an International Standard, they asked the ANS to address two areas in their next review. The two areas where:

- Internationalisation
- Requirements for embedded systems programmed in Forth

It was agreed at euroForth '97 that we would hold a special workshop on internationalisation to investigate the issues the standard will need to address in order to allow programmers to develop multi-lingual applications.

Conference Organiser	*Conference Chair*
Marina Kern	Dr. Peter Knaggs
C/o m2c	Bournemouth University,
Schauenburger Str. 15	Talbot Campus, Fern Barrow,
D 20095 Hamburg,	Poole. Dorset.
Germany.	UK BH12 5BB
Tel: +49 40 325682-10	Tel: +44 1202 595625
Fax: +49 40 325682-90	Fax: +44 1202 595314
Net: m2c@mail.hamburg.com	Net: pjk@bcs.org.uk

Cost: A discount price (given in brackets) is available for delegates registering before the end of July. Note that all prices are exclusive of VAT (currently 16%).

Resident Delegate DM 790 (DM 720)
conference fee, accommodations, 3 meals a day

Student - Limited openings! DM 400 (DM 340)
conference fee, accommodations, 3 meals a day

Guest DM 380 (DM 340)
accommodations, 3 meals a day

4th Day DM 135 per person
accommodations, meals, exhibition and additional workshop

FREEWARE

Fred Behringer's Transputer Forth package F-TP 1.00 is now available at
ftp://ftp.leo.org/pub/comp/os/dos/programming/forth/transputer/

This is a 32-bit nearly ANS, complete Forth for the T800 for use with the INMOS B004, or compatible, board on an IBM compatible PC. It also works with the T400. The server on the host side is based on Turbo Forth, as are the cross-assembler and the metacompiler. The package is freeware and 900 Kb in ZIP form. This includes a precompiled example of a multisystem (many Forths in one). The actual package is substantially smaller.

For information, send e-mail to:
Fred Behringer
behringe@mathematik.tu-muenchen.de

Mike Hore released Mops version 3.2.

Mops is a public-domain development system for the Mac. It's based on Forth, with extensive OOP extensions, along the lines of Smalltalk. It comes with a class library which gives support for all the normal Mac interface functions. While not as full-featured as PowerPlant or MacApp, say, it's very adequate for the kind of applications which might be developed by one programmer.

Mops is derived from Neon, which was one of the first languages for the Mac that allowed actual development on the Mac itself. It's a close cousin to Yerk, which is a more "conservative" development of Neon, basically aimed at keeping up with later Macs and systems while remaining fully compatible with Neon. Mops is more "radical". It's a omplete reimplementation which compiles native (68K and PowerPC) code instead of the usual Forth threaded variety. It's very fast — about as fast as anything on the Mac in fact. It has a few other improvements over the original Neon, such as multiple inheritance, public ivars and temporary (local) objects.

http://www.netaxs.com/~jayfar/mops.html
ftp://ftp.taygeta.com/pub/Forth/Mops/
(taygeta is the main FIG ftp site)

LEVELS OF MEMBERSHIP

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

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

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

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

FROM THE NET

A Shot in the Foot

Bart Lateur wrote: The major problem with Forth is the fact that it's so damn easy to shoot yourself in the foot. Just accidently do something like `0 @` and you'll get a system crash, on many systems.

Anton Ertl replied: That's a problem of the system. Gforth on Linux gives:

```
0 @
:1
0 @
  ^
Error: Invalid memory address
```

and I end up in the text interpreter (or whatever CATCHes this exception).

By the way, with C you can shoot yourself in the foot in the same way (although a little more verbosely):

```
main()
{
 return *(char *)0;
}
```

Compiling and running this on Linux gives:

```
Segmentation fault (core dumped)
```

and I end up in the shell. Catching this with a signal handler is somewhat more work than using CATCH in Forth.

The Computer Journal

Support for older systems
Hands-on hardware and software
Computing on the Small Scale
Since 1983

Subscriptions
1 year $24 - 2 years $44
All Back Issues available.

TCJ
The Computer Journal
P.O. Box 3900
Citrus Heights, CA 95611-3900
800-424-8825 / 916-722-4970
Fax: 916-722-7480
BBS: 916-722-5799

SPONSORS & BENEFACTORS

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

Corporate Sponsors

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

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

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

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

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

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

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

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

www.kernelforth.com

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

www.theforthsource.com

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

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

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

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

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

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

Individual Benefactors

Everett F. Carter, Jr.	Zvie Liberman
Edward W. Falat	Marty McGowan
Michael Frain	Gary S. Nemeth
Guy Grotke	Marlin Ouverson
John D. Hall	Richard C. Wagner
Guy Kelly	

Twentieth Anniversary of the FORML Conference "Forth Interfaces to the World"

November 20–22, 1998 • Pacific Grove, California

FORML welcomes papers on a variety of Forth-related topics, even those which do not adhere strictly to the published theme. Some theme-related topics of interest, and for which papers are particularly sought, include:

Overcoming the Limits to Growth

Forth in "Foreign" Embedded Environments (e.g., Windows CE, Inferno, pSOS, Vrtx)

Forth and Rapid Application Development (RAD)

Forth on New 32-bit Embedded Chips

Forth in a Windows World

Co-Existing with C

Forth and the Internet/Java

"20/20: Hindsight and Vision" is planned as a two-part evening panel. Part one will offer a look at Forth's history—what worked well and what might have been done differently—and will feature participants who played key roles in Forth's evolution; part two will evaluate Forth's current status and propose courses of action to lead Forth into a stronger position in coming years.

Registration Information

SAVE UP TO 20%

Advance registration required. Complete registration by October 15, 1998 to receive a ten percent discount. FIG members are eligible for an *additional* ten percent discount on any registration fee.

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

Conference attendee in double room	$595
Non-conference guest in same room	$435
Under 18 years old in same room	$225
Conference attendee in single room	$795
Infants under two years in same room—free	

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

The FORML Conference is held at the Asilomar Conference Center, a National Historic Landmark noted for its wooded grounds just yards from Pacific Ocean dunes and tidepools on California's Monterey Peninsula. Lodging and all meals included with conference registration, and spouses and guests of conference participants can join numerous recreational outings and activities.

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

Call for Papers

Please submit the subject of your paper as soon as possible in order to be included in pre-conference publicity. Final titles with abstracts are due by October 1, 1998. Completed papers should be received by November 1 in order to be included in the conference notebooks that are distributed to all attendees.

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