FORTH Dimensions

Volume 7, Number 2

July/August 1985
$2.50

Menus in Forth

Probabilistic Dictionaries

Hacker's LOCKER

Mass Transit Forth

Another Subroutine Technique

FORTH Dimensions

FEATURES

DEPARTMENTS

FORTH Dimensions
Published by the
Forth Interest Group

July/August 1985
Volume VII, Number 2

Editor
Marlin Ouverson

Production
Cynthia Lawson

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

Subscription to *Forth Dimensions* is free with membership in the Forth Interest Group at $15.00 per year ($27.00 foreign air). For membership, change of address and to submit material for publication, the address is: Forth Interest Group, P.O. Box 8231, San Jose, California 95155.

Symbol Table

Simple; introductory tutorials and simple applications of Forth.

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

Advanced; requiring study and a thorough understanding of Forth.

83 — Code and examples conform to Forth-83 standard.

79 — Code and examples conform to Forth-79 standard.

fig — Code and examples conform to fig-FORTH.

FORML — Deals with new proposals and modifications to standard Forth systems.

Letters to the Editor

Standard Bearers

Dear Mr. Ouverson:

The Financial Services Division of Computone Systems has developed an applications language specific to insurance using the Forth-83 Standard. We think the remarkable thing about our language is that it runs on 8086/8088, 6502 and Z80-based machines. There is a lot of talk about languages being transportable, but we found that most of it really is nothing but talk. Without the new standard, it is not likely that we could have accomplished the level of transportability that we have.

Computone has been developing software in Forth for the past two and one-half years. Initially, Forth was dictated by the machine we chose to market. We began our support of additional machines by purchasing Forth packages from outside vendors.

One fact became clear early in our development phase: there was no generally accepted standard and little continuity in vocabularies and functions among the Forth versions we were working with. These differences between versions prevented our applications code from being as transportable as we needed. Eventually, we had to decide whether or not to continue our development exclusively in Forth.

When evaluating other languages as possible substitutes for Forth, we found that several of them offered more complete development systems and more liberal licensing agreements. Many of the packages offered better speed and more thorough floating-point tools. On the other hand, there was still no language available that we could alter so that the applications code would be identical on all the machines we were supporting. We found that the better-supported systems were expensive and had their own drawbacks: link/compile time, non-standard extensions, speed or size limitations and highly variable support for machine-specific devices, I/O, etc.

Forth was one language that could be altered at the machine level. If the right version could be found for each processor we were supporting, adjustments could be made to the kernel via the metacompiler to develop the multi-machine applications language we needed.

After some investigation, we found that several individuals had produced public-domain versions based on the new Forth-83 Standard. These versions were similar in philosophy and came complete with the source for their compilers, and had most of what we needed for development.

Using the public-domain versions, we were able to extend the machine-specific code and duplicate the current language that our applications team was using. We found that by simply transferring the high-level code from one machine to another and recompiling, our applications would run on virtually all our machines. We eliminated the eventual need for retraining our staff and extended the useful life of the tools and applications we had developed.

Ultimately, our decision to develop exclusively in Forth allowed us to cut our programming costs and development time drastically. We were able to simplify software development by reducing the number of systems we had to learn and use. We increased productivity by reducing the number of application variations that had to be developed and supported. We were able to deliver the software faster and with far fewer problems. The final result was an increase in our bottom line, and after all, the real value of a language is in how it impacts profits.

We are convinced that without the 83 standard, our integration of language and systems could not have been achieved. The standard proved to be a rallying point from which individuals were willing to work to produce systems which were compatible over a wide range of machines and available to the Forth community at low cost. The presence of a standard for high-level code only serves to make the language more transportable and useful to developers who work with an ever-increasing number of machines. The presence of standards within the community need not curtail the adaptability of the language, but rather serves as a bridge to new solutions, tools and ideas. The new standard and its wide acceptance will help Forth gain the industry-wide credibility it deserves.

Sincerely,

Michael D. Pollard
Donald S. Schrader
Computone Systems
Atlanta, Georgia

Hore Today, EXITed Tomorrow

Dear FIG:

Hore (*Forth Dimensions* VI/6) provides a solution to the so-called "double-loop-exit" problem, as it has been referred to in other quarters. His solution entails the derivation of an alternative **DO LOOP** structure.

Screen 1 shows yet another alternative. In this solution, the system-dependent word **DROP.LOOP** is created which, when executed, allows a normal **EXIT** (given as **;S** in the example, as it is in a fig-FORTH dialect).

DROP.LOOP does what the sequence **R> R> 2DROP** would do if it were included at the equivalent point in the definition of **LOOK.UP**.

For those unfamiliar with the use of **EXIT**, it forces the end of the word, from anywhere in the word — with the notable exception of within a **DO LOOP**. **DO LOOP**s typically leave two or more values on the return stack. Since **EXIT** makes use of the value which sits below those two or more values, we have to dispose of them before we can exit. A little brute force (trial and error) experimentation will show you how your system works. The only other element you may need to take care of would be a Forth-83 implementation of **DO LOOP**, which sets a flag to indicate being in a loop. Your system probably doesn't do this, but the flag should be indicated in your user variables if it does.

The objection raised to the kind of maneuver presented here typically comes from the Wirth school of programming — the solution used does not have a "common end." And, so the argument goes, without a common end, another person reading a long program will get lost and be unable to follow the flow of the program. Arguing from first principles, a la Dijkstra, it sounds sound.

Evidence, however, has a way of modifying principle. **LOOK.UP** is, in fact, totally unambiguous. A stranger to the code (but not to Forth, of course) would understand immediately the action of the word. The difference which matters here is the size of the word: when the program is trivially small and program elements are self-descriptive, it is demonstrably good programming for a word with two distinct outcomes to have two distinct ends.

A key distinction is whether the program is trivially small and is made of self-descriptive program elements, whose own elements have the same characteristics. Which, as I think about it, isn't a bad description of what a Forth program should — and can — be.

I have included screen 2 to show some more uses of those interesting words **COMPILE**, **[COMPILE]** and **IMMEDIATE**. It was my fifth time through *Starting Forth* before I began to grasp them. Useable examples sometimes teach better than either words or pictures.

The word **DO+** saves a little typing and eliminates the clutter of all those **OVER** + **SWAP** sequences we use with a range of addresses. **LOOP.EXIT** performs the functions of **DROP.LOOP** and **EXIT**, as the F83 word does. I typically keep them separate in my programs, as I have found times when program flow is clearer if I do.

Regards,

Henry J. Fay
Cazenovia, New York

Missing Mathquiz

Dear Marlin,

I'd like to look at the Mathquiz program on pages 13-14 of *Forth*

```
SCR # 1
\ DROP.LOOP EXAMPLE                    HES 64FORTH    HJF 01MAY85

: DROP.LOOP
   COMPILE R>  COMPILE R>  \ BRING LOOP INDICES BACK HOME
   COMPILE 2DROP   \ GET RID OF THEM
   ; IMMEDIATE   \ NATURALLY

1 VARIABLE TABLE 2 , 3 , 4 , 5 ,

: LOOK.UP ( VALUE/TABLE-ADDR/#ENTRIES --- ADDR' OR 0 )
   OVER + SWAP
   DO
    DUP I @ =
    IF  DROP I  DROP.LOOP  ;S  ENDIF
   LOOP
   FALSE ;

SCR # 2
\ IMMEDIATE FUN                        HES 64FORTH    HJF 01MAY85

: LOOP.EXIT
   [COMPILE] DROP.LOOP  COMPILE ;S  ; IMMEDIATE

: DO+ ( START-ADDR/#ENTRIES --- )
   COMPILE OVER  COMPILE +  COMPILE SWAP
   [COMPILE] DO  ; IMMEDIATE
```

Mathquiz

```
SCR #72
( PROGRAM VARIABLES                                LDM 08/84 )
VARIABLE CHOICE    1 CHOICE ! ( Get and store players choice # )
VARIABLE RESULT    0 RESULT ! ( Store result for *, +, &- )
VARIABLE SCORE    0 SCORE ! ( Maintain players # of correct ans )
VARIABLE LEVEL    0 LEVEL ! ( Store difficulty level )


( Delay routines                                   LDM 08/84 )

: DELAY    15000 0 DO LOOP ; ( Null loop time delay )

: 2DELAY    25000 0 DO LOOP ; ( Longer time delay )

SCR #73
( TITLE PAGE--MATH-QUIZ )
: STAR    42 EMIT ;
: STARLINE    40 0 DO STAR LOOP ;
: BLANKLINE    STAR 38 SPACES STAR ;
: INFO-L1    STAR 14 SPACES ." MATH QUIZ "
    14 SPACES STAR ;
: INFO-L2    STAR 13 SPACES ." VERSION 1.3 "
    13 SPACES STAR ;
: INFO-L3    STAR 18 SPACES ." BY"
    18 SPACES STAR ;
: INFO-L4    STAR 12 SPACES ." LYLE D. MORTON"
    12 SPACES STAR ;
: MATHTITLE    PAGE STARLINE BLANKLINE
    BLANKLINE INFO-L1 BLANKLINE INFO-L2
    BLANKLINE BLANKLINE INFO-L3 BLANKLINE
    INFO-L4 BLANKLINE BLANKLINE STARLINE CR CR ;

SCR #74
( INSTRUCTIONS FOR MATH-QUIZ )
: INSTRUCT    3 SPACES ." HELLO.  YOU "
    ." ARE ENTERING THE WORLD " CR
    ." OF MATH-QUIZ.  PLEASE ENJOY "
    ." YOURSELF!" CR ." BE ADVISED "
    INVERSE ." THE UPPER LEVELS MAY REQUIRE"
    CR ." A PENCIL AND PAPER.  "
    NORMAL ." GOOD LUCK! " ;
```

Continued

Dimensions (VI/6) but some of the screens seem to be missing. Words like **GETNAME** and **ANSWER** in the text, and many of the words shown in screen 83 are not defined. Could I get a copy of the full set of screens? I'd like to set up the program for my wife's school kids and I'd like to study it to learn something about keyboard input.

Klaus Schleisiek has picked a wonderful place as site of the euroFORML conference. I've been in Heilbronn and through the Neckar valley a couple of times. I'll be at the European FORML conference for sure.

Best!

Nathaniel Grossman
Los Angeles, California

Editor's note: Sincere apologies go to our readers and to author Lyle D. Morton who wrote "Mathquiz." In our issue VI/6 we unintentionally printed only four screens (numbered 71, 79, 80 and 83). The accompanying screens 72-78, 81 and 82 are published here to complete the program.

```
SCR #75
( MATH QUIZ --- ADDITION MODULE )

VARIABLE ADD1   0 ADD1 ! ( addend # 1 )
VARIABLE ADD2   0 ADD2 ! ( addend # 2 )
: ANSWER  QUERY BL WORD NUMBER DROP ; ( gets players answer )
: ADDITION    ( MAKE PROB & GET ANS)
     LEVEL @ CHOOSE ADD1 ! ( get difficulty level and )
     LEVEL @ CHOOSE ADD2 ! ( use CHOOSE to select addends )
     CR ADD1 @ . ."  + "
     ADD2 @ . ."  = " ( 8 & 9 format problem )
     ANSWER
     ADD1 @ ADD2 @ + DUP RESULT ! ( add the addends & store it )
  = IF ." CORRECT " 1 SCORE +!
      ELSE CR ." WRONG, THE CORRECT "
       ." ANSWER IS " RESULT @ . THEN ;


SCR #76
( SUBTRACTION MODULE )

VARIABLE SUB1   0 SUB1 ! ( minuend )
VARIABLE SUB2   0 SUB2 ! ( subtrahend )
: SUBTRACTION  ( MAKE PROB-GET ANS)
    LEVEL @ DUP CHOOSE + ( add random number to LEVEL )
    SUB1 !        ( to make a minuend > subtrahend )
    LEVEL @ CHOOSE SUB2 ! ( choose subtrahend )
    CR SUB1 @ . ."  - "
    SUB2 @ . ."  = " ( 8 & 9 format problem )
    ANSWER
    SUB1 @ SUB2 @ - DUP RESULT ! ( get difference and store )
    = IF ."  CORRECT " 1 SCORE +!
    ELSE CR ." WRONG, THE CORRECT"
    ."  ANSWER IS " RESULT @
    . THEN ;


SCR #77
( MULTIPLICATION MODULE )

VARIABLE MULT1   0 MULT1 ! ( multiplicand )
VARIABLE MULT2   0 MULT2 ! ( multiplier )
: MULTIPLICATION  ( get problem and players answer )
    LEVEL @ CHOOSE MULT1 ! ( randomly choose multiplicand )
    LEVEL @ CHOOSE MULT2 ! ( randomly choose multiplier )
    CR MULT1 @ . ." * "
    MULT2 @ . ." = " ( 7 & 8 format the problem )
    ANSWER
    MULT1 @ MULT2 @ * DUP RESULT ! ( get product and store )
    = IF ." CORRECT! " 1 SCORE +!
    ELSE CR ." WRONG, THE CORRECT"
    ."  ANSWER IS " RESULT @ .
    THEN ;


SCR #78
( DIVISION MODULE )

VARIABLE DIVI1   0 DIVI1 ! ( divisor )
VARIABLE DIVI2   0 DIVI2 ! ( quotient )
: DIVISION    ( GET PROB & ANSW )
    LEVEL @ CHOOSE 1+ DUP DIVI1 ! ( get > 0 divisor )
    LEVEL @ CHOOSE 1+ DUP DIVI2 ! ( get > 0 quotient )
    * RESULT ! CR ( develop dividend )
    RESULT @ . ." / "
    DIVI1 @ . ." = " ( 8 & 9 format the problem )
    ANSWER DIVI2 @ ( compare quotient with player's ans )
    = IF ." CORRECT " 1 SCORE +!
    ELSE CR ." WRONG, THE CORRECT"
    ."  ANSWER IS " DIVI2 @ .
    THEN ;
```

Continued

You Screen, I Scream

Dear Marlin,

The Forthodoxy commands that we Forth our programs into screens, the "natural unit of thought." As a confirmed heterodox, I think it is time to examine what screens offer and do not offer, in contrast to their alternative.

Right off, let's admit that screens are not a natural unit of anything except computer memory or CRT display. The mind doesn't inherently work with 1024-byte units — certainly not when any number of those bytes can be blank. To the extent that screens keep us 1K-bound, they make it difficult to deal with our code at flexible levels of detail and generality.

To appreciate how disfunctional screens are, note their perfect analogy to the numbered lines of BASIC. Like line numbers, screen numbers are inherently meaningless, presenting housekeeping chores and working against self-documentation. The process of copying, moving and inserting screens is tedious (even hazardous),

```
SCR #81
( GREETING MODULE )

CREATE   NAME 40 ALLOT
: GREET   PAGE ." WELCOME TO "
   ." MATHQUIZ VERSION 1.3 !" ;
: GETNAME   CR ." ENTER YOUR FIRST"
   ."  NAME. " CR QUERY 1 TEXT ( get players name )
    PAD NAME 40 CMOVE ; ( move string from PAD to NAME )
: PLAYER   NAME 40 -TRAILING TYPE ;
: REGREET   PAGE PLAYER ."  , YOU "
   ." MAY CHOOSE FROM THE " CR
   ." FOLLOWING MENU : " CR ;
: COMMENCE   GREET GETNAME  ;



SCR #82
( LOOP & SCORE MODULE )
: DISPLAY   CR PLAYER ." , YOU "
   ." WERE CORRECT " SCORE @ DUP
    2DUP . ." TIMES " CR ." OUT OF 10."
    INVERSE 8 >  IF ." VERY GOOD!" 2DROP ELSE
    6 >  IF ." NOT BAD!" DROP ELSE
    ." BETTER WORK HARDER!" THEN
    THEN NORMAL ;
: 10-ADD    0 SCORE ! 10 0 DO ( these set up blocks of 10 )
    ADDITION LOOP DISPLAY  ;
: 10-SUB    0 SCORE ! 10 0 DO
    SUBTRACTION LOOP DISPLAY  ; ( and zero the score at the )
: 10-MULT    0 SCORE ! 10 0 DO
    MULTIPLICATION LOOP DISPLAY ; ( beginning of each block )
: 10-DIV    0 SCORE ! 10 0 DO
    DIVISION LOOP DISPLAY  ;
```

hence we create "spaghetti loads" that make source code harder to follow. Indeed, the **LOAD** command is BASIC's GOTO expressed at a different level.

In addition to spaghetti loads, screen boundaries encourage "meatball code." Long, horizontally formatted, uncommented definitions are a natural result of a screen boundary that takes effort to cross — and the fact that blank areas get wasted. By the way, index lines — manually added redundance — are another space waster.

So far, I've said that screens work against flexibility, space economy and self-documentation. Readability suffers from spaghetti and meatballs, plus excess baggage (screen number in decimal and hex, initials, date, index line and line numbers). Screen editors themselves are horribly weak, so even interactivity can suffer. There is hardly a virtue claimed by Forth that screens do not work against.

When resources are limited, screens offer an easily implemented, interactive way to write programs in an incremental fashion. Fortunately, resources are no longer *that* limited, and file systems with powerful editors have *already* been implemented. The most natural, least restrictive way to produce code is with good text editors or word processors. I won't dwell on their advantages over Forth editors. To anyone who has tried them, the difference is obvious (in New York, "FIG edit" means "forget it"!).

Besides offering greater power, ease and flexibility, a real editor conserves disk space (I know — big deal) while encouraging commenting and high readability. With a good file system, it offers self-documenting file names for code modules of whatever size the programmer chooses.

Any problems? One is indexing; without it, well-organized code in well-named files is important (a good idea in any case). Another problem is navigation between Forth and the editor; to which fast operating systems, multitasking, desktop utilities or non-screen editors within Forth are all solutions. Lastly, compilation speed may suffer, depending on how politely Forth relates to its host; the trend is toward politeness, which eliminates the problem.

Editorial

Journey to the East

We recently returned from the annual Rochester Forth Conference. The lively event was sponsored by the Institute for Applied Forth Research in cooperation with the Laboratory for Laser Energetics *and the IEEE Computer Society*. That says something interesting about the times if anything does. The directors and staff of the Forth Institute assembled a stimulating program that is reviewed elsewhere in this issue in necessarily brief terms — there were about sixty presentors.

In this issue we make good on promises past. Due to a last-minute error, our last cover promised "Menus in Forth" but it was nowhere to be found inside — the article had been rescheduled. This time you'll find it, honest! Also included herein are screens to accompany Morton's "Mathquiz." Thanks to the readers who wrote to request them. As a side note, columnists John Hall and Henry Laxen are absent, at least for the time being.

Add to the list of FIG membership benefits a new one: major medical insurance. In the U.S. only, FIG members, their families and possibly their employees are now eligible for group rates. Current members will be receiving notice in the mail with specifics about the health-care plan. Those of you who work independently or who run a small business may be particularly interested.

And, finally, we would like to welcome five new chapters of the Forth Interest Group. Members are now holding official meetings and other Forth-related activities at MAD Apple FIG Chapter, Madison, Wisconsin; Cache Forth FIG Chapter, Oak Park, Illinois; Permian Basin FIG Chapter, Odessa, Texas; Japan FIG Chapter, Tokyo and Kyoto, Japan; and Rockland County FIG Chapter, Pearl River, New York. Welcome to each group, may your meetings prosper. For a complete listing of all international FIG Chapters, turn to the back pages of this magazine.

—Marlin Ouverson
Editor

I've used a file-and-editor system for three years, and in spite of dire warnings, the only lightning I've been struck by is my own programming speed. I think it is vital that we move toward elimination of the restrictions represented by screens. Unless it can adapt to the progress around it, the Forth may not always be with us.

Sincerely,

Laughing Water
Helena, Montana

Division Floors Him

Dear Marlin,

I have never been one to pride myself on an understanding of divisional mathematics, and the debates between floored-to-zero versus floored-to-negative-infinity have escaped me. Well, you can imagine my delight when I recently had occasion to use division in a situation that illustrates the usefulness of the latter method.

In an appointment scheduler, the day has been divided into five periods. It is necessary to access prior and later periods. A single word **periodaway** can handle the necessary calculations. **adjustperiod#** uses **/MOD** to calculate the number of days crossed and resets **period#**. This simple math is all that is required for any number of periods. The word **dayinterval** adjusts the date for a change of n days. It is dependent on the calendar module used and serves no purpose in this discussion.

The periods used are: 0 = dawn, 1 = morning, 2 = afternoon, 3 = evening and 4 = night. If one begins with morning (setting **period#** to 1), then **-7 periodaway** resets **period#** to -6 and **-6 5 /MOD** returns 4 -2. This adjusts the date two back and resets the **period#** to 4 (night), as it should be. What could be simpler?

I raise my hat to the standards team on this one.

Zaffar Essak, M.D.
Vancouver, British Columbia
Canada

```
Block # 1

( periodaway              Appointments            850512z)

\ dayinterval ( n--)      Adjusts date by n days.

0 VALUE period#

: adjustperiod# ( --)
   period# 5 /MOD dayinterval \ floored to negative
   TO period# ;               \   infinity ideal !!

: periodaway ( n--) AT period# +!
   period# 0 4 WITHIN 0=
     IF adjustperiod# THEN ;

EXIT
```

A Case of Brackets

Editor,

Henry Laxen's article in your March/April issue (VI/6) was very interesting for his discussion of using] as a callable compiler. Looking into just how this works in the Perry/Laxen F83 system, described in C.H. Ting's book *Inside F83*, I find a difficulty. When reading from the terminal, the] in F83 compiles only one line of text. Colon definitions in F83 can have multiple lines because compilation is continued by repeated] calls in **QUIT**. No return into the definition of colon is made, but none is necessary. Yet, such a return into the defining word is just what is wanted from] as it is used in Henry's **CASE**, for example.

This restriction seems inherent in trying to use] as a callable compiler instead of some other word. The] is used not just to begin compilation, but is used with [to enclose an interpreted expression embedded in the source text of a compiled definition. Such embedded expressions really require a single interpreter vectored back and forth from compiling to interpreting by brackets, rather than calling and returning. This interpreter, actually the word **INTERPRET**, already does everything Henry's] does, if called with state set to compile.

Like Henry's], though, **INTERPRET** only compiles one line from the terminal. Rather than change], therefore, I would want to change **INTERPRET**. The repeated loop over **QUERY**... **INTERPRET** in **QUIT** should be moved into **INTERPRET** so that a single call will process all the lines. The loop must repeat only when input comes from the terminal instead of disk. The null word executed at the end of a line should return into this loop, while exiting from **INTERPRET** is done by ; at the end of a definition.

One way to do this is by having **INTERPRET** call a component word **(INTERPRET)** to process each line. Null can then be simply **EXIT**, as in fig-FORTH; it exits **(INTERPRET)** at the end of a line. Semicolon can be defined simply as an exit one more level, to the caller of **INTERPRET**, which will be a defining word. The other work now done by semicolon — compiling **EXIT**, etc. to finish a colon definition — can actually be done in colon. Semicolon then can be used to terminate any compiler construct, not just those using colon. Henry's version of semicolon lacks this generality.

Henry described the compiling] as an outgrowth of Forth-83, which eliminates state-smart words. (I still like state-smart words, but they really are irrelevent here. Besides, Forth-83 does not define] as processing text; like Forth-79, it defines both brackets vaguely, as setting state so that text is "subsequently compiled" or "subsequently interpreted." Code using Henry's] therefore may not work on some quite standard systems.

On another subject, credit should be given to D. Val Schorre for his paper on "Adding Modules to Forth" in the proceedings of the 1980 FORML Conference. He presented the same technique described by Carol Pruitt in your March/April issue under the title "Local Definitions."

Sincerely yours,

George Lyons
Jersey City, New Jersey

NEW BOOKS - "FORTH GUIDE" by Haydon. An exploration of the intricacies of MVP-FORTH $20.00
"FORTH Applications" by Weber $13.00

Ask the Doctor

William F. Ragsdale
Hayward, California

"Ask the Doctor" is Forth Dimensions' *health maintenance organization devoted to helping you understand and use Forth. Questions of a problem-solving nature, on locating references, or just regarding contemporary techniques are most appropriate. When needed, your good doctor will call in specialists. Published letters will receive a preprint of the column as a direct reply.*

This month your faithful practitioner deviates from his usual format. In the last three issues, we examined aids to learning Forth. Interest shifts this month to news and commentary on the latest events impacting the Forth community. The first stop on our morning rounds occurs at the Forth nursery, otherwise known as the Chip Hatching Department.

Novix NC4000 Processor

The hottest topic of interest in the Forth world must be the Novix NC4000P Processor, Charles Moore's embodiment of Forth in silicon. This sizzler of a processor gives about 10,000,000 Forth instructions per second (10 mips). Up to four Forth instructions (like **DUP I** +) can execute at each 125 ns clock cycle. A call takes one clock cycle and a return takes none! This rewrites the book on software, obviating interest in linear code and macros.

To bring this topic into focus, we peek behind the scenes to witness a bit of the genesis of the Forth processor. The courtship began in October of 1980. The moment of conception was on January 19, 1981.

First things first. Forth, Inc. had decided to expand its Board of Directors. Bill and Anne Ragsdale entertained Mr. John Peers at dinner in Palo Alto in October 1980, with the ultimate result of Mr. Peers being invited to join the Board. John Peers was the founder of Logical Machines, which had an innovative, extensible language called Adam. He was involved in robotics and in the quest for aware machines.

The spark that set the Forth processor development into motion came just two months later. We owe a debt of gratitude to Christine Colburn (president of Creative Solutions), who gave a $1000 birthday present to husband Don. Don had been examining computer architectures and had developed a passion to see the advantages of Forth directly applied in silicon.

Don thus funded a one-day, project-organizing session with Charles Moore, Bill Ragsdale and a chip design consultant. This was the spark needed for Chuck to confirm that others were seriously in support of his idea for a Forth processor.

John Peers saw the beauty of approaching Forth from several levels. He founded Technology Industries in March 1981, with the encouragement of Chuck Moore and Elizabeth Rather of Forth, Inc. As the story unfolds, Forth Inc. merged into Technology Industries in August 1981. The charter of Technology Industries was to be an umbrella for three divisions of Forth: hardware, software and applications. Unfortunately, the full scope of this plan was not reached and the merger was rescinded in October 1982.

The chip development still continued, funded with the limited resources of Technology Industries. Charles Moore demonstrated a color simulation of the processor in March 1983. A funding partner, Sysorex International, became interested in July, and by March of 1984 the Novix partnership commenced operation. John Golden transferred from Technology Industries as general manager, and continues in the capacity today.

Silicon generation was almost anti-climatic. It took four years and a million dollars to get to the detail design stage. It took just seven months and seven hundred thousand dollars to realize the dream of an operational Forth processor in silicon.

EDN magazine had the privilege of introducing the NS4000 in their March 1985 cover article. John Golden, guiding light of Novix, says, "The *EDN* article put us on the map. Three hundred inquiries were developed. We found that Forth has friends throughout industry just waiting to show their management the opportunities."

The first production run of development boards is nearly sold out. Mr. Golden says shipments will commence in the latter part of July with full documentation and developmental software (by Gregg Bailey). Production units were demonstrated at the Silicon Valley FIG Chapter meeting and at the Rochester Forth Conference. Purchasers' names read like a who's who of technology. There are automobile manufacturers, a computer media firm and a computer company that is into fruit. The processor is even being considered as a controller co-processor for a recent thirty-two-bit family. When queried on their motive, a spokesman replied, "We see this as a way to get a leg up on the rest of the world."

Forth and Voice Mail

FIG's own John Cassady is getting no end of attention in the technical and popular press. He was recently "discovered" by *Infoworld* columnist John Dvorak. Mr. Dvorak, you will recall, is fondly remembered for his quote that he has never seen a credible application done in Forth.

He has met his match, as John Cassady recently introduced the Dyna-Com Voice Mailbox System. It is programmed in Forth and is a honey of a product. Dvorak devoted an entire column in the *San Francisco Chronicle* to Cassady's product and then gave it another hit in his *Infoworld* column. You may try the system and leave remarks for John Dvorak at his voice mailbox by calling (415) 763-2002. When the system answers, tone in 454.

The system supports twenty-five users, each with his own addressable "box." Callers leave messages which may be interactively reviewed by the boxholder. But each boxholder may grant up to thirty of his friends their own guest mail boxes for replies! Imagine 750 people with selective use of digitally controlled voice communications. The box is an IBM PC look-alike processor board with a hard disk, speech digitizer and Touch-Tone signalling. From the outside, it looks like just another industrial box. But on the inside, the VMS has a heart of pure Forth. John Cassady was previously best known as the implementor of fig-FORTH on the 8080. He published the 8080 assembler in widest use and is the author of the book *Metaforth*.

The DynaCom system is to be demonstrated at the Silicon Valley FIG Chapter meeting on July 28. Too bad our publication date means you will receive this issue of *Forth Dimensions* the following week! FIG even has its own voice mail box. For information on the next Silicon Valley Chapter meeting call (415) 763-2002 and tone in 44414. For a demonstration, key in 564.

The Rochester Conference

The Fifth Rochester Forth Conference continues to recieve rave reviews from attendees. During early June, over 180 people were hosted by Thea Martin and Larry Forsley. Participants traveled from all over the U.S., Canada, the Netherlands, Germany and the U.K. The program featured ten papers on Forth computers, with three from Novix. Novix' Bob Murphy packed the house with the best rundown yet on the internals of the NC4000.

Major firms are using Forth and appear to be making a significant commitment in their project work. High-visibility attendees at Rochester included Standard Oil, Bell Canada and the GM Delco Division.

The working group on standards, presided over by Mahlon Kelly (no relation to Guy Kelly, Forth Standards Team chairman), got into the topic of transitions from Forth-79 to Forth-83. A significant attitude was expressed about the burden and value of shifting to Forth-83 from an earlier dialect. Some opinion was offered that system and application changes weren't justified by the small perceived benefit.

Switch or Fight?

Forth-79's five-year lead has finally produced a comfortable user base. Working group members asked "Why update to Forth-83? Is the effort worth the improvement? What will be the common model of a full implemention?"

fig-FORTH was the common model prior to Forth-79. MVP-Forth carried the FIG model forward, and has become the *de facto* model for Forth-79. F83, the Laxen-Perry model, is only now coming into its own as the leading contender for the Forth-83 Model sweepstakes. Your faithful servant offers a few opinions. (Who said, "Anybody can have the facts, but it takes real character to have opinions"?)

First, it takes four years for a language standard to become accepted and popular. Do you remember the hue and cry about Forth-79? It was supposed to be the worst disease since the plague. Refer to your back issues of *Forth Dimensions* to check it out. We are two-and-a-half years into Forth-83. It will catch full swing in another eighteen months. Just wait!

Second, the Forth Interest Group must abandon fig-FORTH. The system design and listings are four years behind the times, unsupported, impossible for the novice to self-install, and at cross purposes to the learning process. (The only viable replacement for the revenue that would be lost is a disk-based applications library, but this has its own set of complications.)

Third, some complaining occurred at Rochester that the shift to Forth-83 was being pressured upon those complaining. This is a touchy topic. Some standards are required. Building code standards, for example, are enforced by local governments. But Forth usage remains voluntary. The Forth Standards Team went to special lengths to make this clear. Read the copyright notice on Forth-83, to wit:

"The existence of a Forth Standard does not in any respect preclude anyone . . . from implementing, marketing, purchasing or using products, processes, or procedures not conforming to the Standard."

The complainers probably are reacting to their perception of the transition around them, and divert attention to the ogre of enforced change when trying to cling to the past. My understanding of the standards team's expression is: if your application is done in a prior standard, let it run. If you are considering new effort, and want it to have the longest life, use the latest standard available. As a professional, you have a choice.

Next Issue

The clinic will resume addressing reader inquires in the next issue. Machine-specific problems such as how to install fig-FORTH on a VAX under UNIX aren't of general interest. Questions should revolve around Forth, in the manner of, "When <**BUILDS** is replaced with **CREATE**, does it have to be preceded by **COMPILE**?" Until next time, I remain your obedient servant.

For More Information

For information on the NC4000P processor, contact John Golden at Novix, 10590 N. Tantau Ave., Cupertino, California 95014; (408) 996-9363.

For data on the DynaCom VMS VoiceStar, contact John Cassady at 339-15th Street, Oakland, California 94612; (415) 763-6636.

MVP-Forth is distributed by Mountain View Press, Post Office Box 4656, Mountain View, California 94040; (415) 961-4103.

F83, the Laxen & Perry Model, is available on diskette at $25 for IBM PC, CP/M 8080 and 68000, from No Visible Support Software, P.O. Box 1344, 2000 Center Street, Berkeley, California 94704.

The Rochester Forth Conference is held in June, organized by the Institute for Applied Forth Research, Inc., 70 Elmwood Avenue, Rochester, New York, 14611; (716) 235-0168.

About the Author

Bill Ragsdale has been using Forth since 1977 for personal and business projects. He is married to Anne, and they have children Mary, three, and Michael, one. The family recently completed a Caribbean cruise with 250 magicians. The ship offered computer classes on twelve IBM PCs, which Bill chose to skip. However, he was pleased to find MVP-Forth in the ship's software library.

Menus in Forth

Frans Van Duinen
Toronto, Ontario

Programs that display menus to show the various options available can be made very easy to use, especially for the novice user. This is particularly significant where the user gets minimal training and documentation (if any at all), such as with public-domain software.

The menu program shows all options available at any specific point, i.e., now that you've selected option A, these are the available sub-options... Menus also tend to lead the programmer along in how he organizes his material. Hierarchical structuring is a powerful way of organizing one's thinking: Think about the most important items first and ignore the details, then go down to a lower level and concentrate on the aspects of only one item, etc.

Menus limit the number of options presented at any one time. With a single item per line, and after allowing for headings and such, you typically put on the screen no more than eight to fifteen items (and eight is better), with an entire line to describe each item — more if needed. That, too, makes for easier use, i.e., better programs.

Even the expert user, who does not need all this detail, can be accommodated. Simply implement a key-ahead facility. If the user knows he wants item A on this menu, then 1 on the next sub-menu and then sub-sub-item C, let him key A1C. Then simply skip directly from the A-level menu down to whatever it is that A1C means or does.

Building Menus

Three things are involved in menus: (1) Display the text of the menu, (2) accept the menu option selected and (3) execute the code corresponding to that selection.

Those are must-haves. For good measure, we'll also throw in (4) a Help facility and (5) the ability to back out of a menu without selecting any of its items.

```
.\ Menu Utility words - version 1                       FVD28Apr84
  VARIABLE M#              \ No of menu item & max
: .RDEC BASE @ DECIMAL .R BASE !  ; \ Display decimal no
: 0M#  M# OFF ;            \ Reset menu no to zero
: (.#")   (S  -> )         \ Define numbered menu entry
  1 M# +!  M# @ 3 .RDEC ." - "    \ Show item no
  R> COUNT 2DUP + >R       \ Step Return addr past text following
  TYPE  CR   ;             \ & display
: .#"             (S  ->  )  \ Compile numbered menu item
    COMPILE (.#") ,"    ;  IMMEDIATE
: (.#0")  (S  ->  )        \ Show menu entry #0
  0  @ 3 .RDEC ." - "       \ Show item no zero
  R> COUNT 2DUP + >R       \ Step past text following
  TYPE  CR   ;             \ & display
: .#0"             (S  ->  )  \ Compile menu item zero
    COMPILE (.#") ,"    ;  IMMEDIATE


.\ Menu Utility words  2 -  Version 1                   FVD28Apr84
: (.M")   (S  ->  )        \ Show un-numbered menu entry
  R> COUNT 2DUP + >R       \ Step past text following
  TYPE  CR   ;             \ & display
: .M" STATE @  IF          \ Compiling?
    COMPILE (.M") ,"
  ELSE
    ASCII " WORD COUNT TYPE  CR  \ Show item text
  ENDIF  ;   IMMEDIATE
: M?    (S  max min -> no  )  \ Get menu selection (0-9 max)
  ASCII 0 +  SWAP  ASCII 0 + \ Set max & min as digits
  BEGIN  KEY  >R              \ Get selection & save
                              \ Infuture Check for ? or Esc
    2DUP R@  -ROT BETWEEN  0=  \ Validate selection
  WHILE  R> DROP  REPEAT      \ While not valid
  2DROP  R> ASCII 0 -  ;      \ Return number
```

Figure One
First Version of Menu Word

```
.\ Menus - Version 1 - example                          FVD28Apr84
: .MHDR  DARK  .M" MXSET Program" ;
: .MNMEN        (S -> ) \ Main Menu
  0M#  .MHDR
  .M" -Main Menu-"  CR
  .#" Set printer"     CR
  .#" Double density"    CR
  .#0" Exit program" CR
  m# @  0 M?             \ Get menu selection
;
\ Must be completed with CASE
```

Figure Two
Version One Example

I've included three versions of menu words. Each successive version is more elaborate and somewhat more difficult to understand. This progression from simple to reasonably complete is how I developed the various ideas. (This code is available on various RCP/M bulletin boards. The file **MXSET.BLK** is a program written in F83 to configure the Epson printer. Note that it is not a complete program, but rather a "Study in Menus." There is also **EDITOR.BLK**, a WordStar-compatible screen editor that I adapted to F83 from the original version published by Laxen in *Dr. Dobb's Journal*, September 1981.)

Figure one shows the first approach. It is concerned primarily with display-


```
 .\ Menu support words - Version 2: .A"  !ANS          FVD28Apr84
: !ANS          (S char  -> )    \ Save allowed response
   PAD DUP C@ 1+ 2DUP  SWAP C! + C!   ; \ Update response list

: (.A")         (S ->  )   \ Display menu item
   R>  COUNT 2DUP +  >R    \ Step past text
   1-  SWAP  DUP  C@ DUP  EMIT \ Display lead char
   !ANS                     \ & save as allowed response
   ."  - "  1+ SWAP TYPE CR ;  \ Display rest

: .A"           (S ->  )   \ Compile menu item
     COMPILE  (.A")  ,"    ;   IMMEDIATE


 .\ Menu response words - (HELP), UKEY                 FVD28Apr84
DEFER  (HELP)          \ Help word for ? response

: UKEY          (S  -> A  )
   KEY  DUP ASCII a  ASCII z BETWEEN
   IF ODF AND  ENDIF   ;  \ Get upper case key

VARIABLE  HELP#             \ Reference no for Help

 .\ Menu support words - Version 2: NEW-MENU           FVD28Apr84
VARIABLE  (M-STK)  0E ALLOT  \ Stack of Menu RP@ & SP@
\ The Menu Stack (M-STK) maintains exit RP@ & SP@ addr for 6
\ levels of menus. This is used by M-EXIT to abort current menu)
: STK-MENU       ( S  ->  )
  R>  RP@  SP@  (M-STK)  DUP @ 2+ +  2!  \ RP@ for calling wd
  4 (M-STK) +!  >R  ;
: USTK-MENU       ( S  -> RP@  SP@  )
  -4 (M-STK) +! (M-STK) DUP @ 2+ +  2@  ;
: NEW-SEL       (S help# ->  )
   0 PAD !             \ Set new selection, not return addr
   HELP# !  ;          \ Set help reference

: NEW-MENU      (S help# ->  )
   NEW-SEL             \ Clear screen & valid responses
   STK-MENU  ;         \ Where to return to on Esc


 .\ Menu - M-EXIT - Version 2                           FVD29Apr84
: END-MENU
   USTK-MENU  2DROP  ;  \ End of menu - clear stack item

: M-EXIT          (S  -> )
   USTK-MENU  OVER RP@ 1 ABORT" Bug in unstacking menus"
   U< ABORT" Menu unstack error"
   SP!  RP!  ;
(S exit current menu & return to calling level menu)

\ Note - Should work but does not.
```

Continued

ing the menu. The word **.#"** compiles the text following it, and the run-time word **(.#")**. The word **(.#")** displays that text. It also displays the menu selection number for that item.

M# is the variable for the menu item number. **.0#"** and **(.0#")** are used for menu item zero, if at all. **M?** accepts a single digit, zero up to the maximum in **M#**. The word **.M"** is used to display an item without a number. It's like **."** but with a built-in **CR**. The **.#"** word is actually more complex than necessary. It does not have to be state dependent. That was done for some other experiments.

This version does nothing to execute the code for the selected option. You have to define a **CASE** construct with words for each option. Version one works, but it did not satisfy me. I did not like the numeric selection (difficult to remember), and I wanted more features.

A Better Version

Enter version two (see figure three). Instead of using a counter, I used **.A"** and **(.A")** — the successors to **.#"** and **(.#")** — to build a list of valid response codes in **PAD**. By convention, the first character of the item text would be displayed separately as the selection character and would be added to the list in **PAD**. The first byte in **PAD** counts the number of entries.

In this version, I also added a Help exit and an escape mechanism. **MENU-SEL** accepts the input and validates it. Valid responses are those in **PAD**, ^C (break) to abort, ? for help and Esc to exit to the calling menu.

We still have the word(s) to display the menu, **MENU-SEL** to get a valid response, and then a **CASE** to execute the selection, based on the index returned by **MENU-SEL**.

Have a look at figure four. This is the main menu for the Epson set-up program. How many items are displayed depends on the variable **GTRAX**. (There are more options on the printer with Graftrax than on the one without.)

The problem is that now we have a variable length list of codes and a fixed

```
 .\ Define menus - Version 2                               FVD28Apr84
: MENU-SEL   ." ? "   (S -> index )
  BEGIN  UKEY  0                  \ Get key & set index
    PAD COUNT OVER +  SWAP        \ Scan allowed responses
    DO  OVER  I C@  =             \ -> char index flag
      ?LEAVE  1+  LOOP            \ -> char index
    PAD C@ 1- OVER <              \ Index past end? - yes
  WHILE DROP                      \ Drop index, check specials
    DUP  3  =  ABORT" Break"      \ Terminate on ^C
    DUP  ASCII  ?  =  IF (HELP)  ELSE
    1B =  IF  M-EXIT  ENDIF       \ exit calling  menu
    BEEP  ENDIF                   \ No BEEP after Help
  REPEAT  SWAP  EMIT  CR  ;       \ -> index
(P Accept input key, & match against acceptable list, returns in
dex. If not matched, abort on ^C, HELP on ?, exit on ESC. Exit
only on valid response.)
```

Figure Three
Enhanced Menu Words

```
: .TITLE  DARK  ."        MXSET - Printer Setup" CR CR ;

: .MNGTR          (S  -> )
  (GTR) @  IF          \ Is this graftrax
    .A" KReset printer to defaults"
    .A" LSet uni-directional print on/off"
    .A" MSet bit 8 handling"
    .A" NSet graphics mode"
    .A" OHome print head"
   ELSE
    .A" KSelect character set"
    ASCII K DUP 2DUP !ANS !ANS !ANS !ANS \ Allign posn
   ENDIF   ;
\ Graftrax dependent features

 .\ Define menus                                           FVD29Apr84
: .MAIN           (S  -> )
  .TITLE   ." Main Menu"  CR CR
  .A" ASet printer type"
  .A" BSelect character font"
  .A" CSet line spacing"
  .A" DSet page size"
  .A" ESet to top of form"
  .A" FSet perforation skip"
  .A" GSet out of paper signal"
  .A" HPrint test"
  .A" IKeyboard to printer"
  .MNGTR                         \ Options specific to GTRAX/not
  .A" ZTo return to CP/M" ;      \ Position dependent

 .\ Menus - MAIN                                           FVD28Apr84
10  CASE:  :MAIN
MX80/100        FONT            SPACING         PAGE
KEYBOARD        BYE             RESET/CHAR      UNI
8BIT            GRAPHIC         ONE-WAY-1       BYE     ;

: MAIN
  0 NEW-MENU  BEGIN  0 NEW-SEL
    .MAIN  1B !ANS     \ Intercept Esc
    MENU-SEL :MAIN
  AGAIN  ;
```

Figure Four
Examples Using Version Two Menu Words

(positional) **CASE** statement. Every item in the menu had better always be in the same spot. The **ELSE** clause in **.MNGTR** enters dummy responses in **PAD** to ensure its length does not change. If Exit is the fifteenth word in the **CASE**, then **MENU-SEL** had better always return a 15 when Exit is selected!

The escape mechanism, too, was troublesome. As a general facility, it should clean up the stack as well as exit. Since we are talking about returning to a calling routine/menu, a simple solution is to keep track of the position on the stack before calling the lower-level routine. Should that lower-level routine abort (error exit, etc.), simply restore the stack pointer to that known value.

I generalized this to cover both stacks, and set up a little menu stack. On this is kept the value for the two regular stack pointers. This is to allow nesting of menus and returning one level at a time. I never did get it to work. It should — the concept is sound — but it is fragile code. Messing with the return stack must be done just right, or it will crash the system.

The Ultimate Version

Fortunately, by this time I had solved the problem. Version three is a little more complicated, but much more readable. It actually works.

In version three, the menu items and the corresponding execution word are combined. There is no **CASE** statement. Have a look at the sample in figure six first. **.A"** is now called **.AX"** and is followed by the name of a Forth word first and then by the text. **(.AX")** now not only puts the response code in **PAD** but also the address of the word that is to be executed should that code be selected. The first byte in **PAD** still counts the entries, which are now three bytes long.

We have removed the positional dependence of the **CASE** statement. **MENU-SELX** gets the answer, validates it and executes the word whose address is stored immediately following. We have also done away with the menu stack. If the option is Esc, do nothing; don't execute any word, merely return to the calling word/menu.

The Help concept is shown here in its more rudimentary form, a word called from **MENU-SEL** when ? is pressed. It uses the variable **HELP#** as set in each menu by **NEW-MENU** or **NEW-SEL**. The number in **HELP#** would indicate what help information is relevant for that specific menu.

The one limitation I am aware of (and have fixed since) is that after a Help message is displayed, the screen should be cleaned up and the menu restored. I've done this by passing the address of the word that displays the menu to **MENU-SELX**, which will re-execute it after Help has been invoked.

There it is. I hope it is of use to others as well. Feel free to use this code for any purpose that is legal and moral.

Acknowledgments

The Forth dialect used is Forth-83 as implemented by Henry Laxen and


```
 .\ Menu support words - Version 3: .AX" !ANSX          FVD28Apr84
: !ANSX        (S addr char  -> )    \ Save allowed response
   PAD COUNT 2DUP 3 + SWAP 1- C!     \ Update count byte
   +  DUP 1+ -ROT ! !  ;
(P !ANSX stores menu char & rtn addr in pad, updates offset)
: (.AX")       (S ->  )          \ Display menu item
   R> DUP @  SWAP 2+             \ -> word, text addr
   COUNT 2DUP +  >R              \ Step return addr past text
   SWAP  DUP  C@                 \ Get lead char
   DUP  EMIT  -ROT  ."  - " \ -> addr char count addr
   1+ SWAP 1-  TYPE CR           \ Display rest  -> word addr, char
   !ANSX   ;                     \ & save as allowed response
(P Displ menu item & store its select. char & rtn addr in PAD)
: .AX"               (S ->  )    \ Compile menu word & text item
     COMPILE  (.AX")             \ leaves (.AX"), word, text
     [COMPILE]  ' ,  ,"   ;      IMMEDIATE

 .\ Define menus  Version  3                            FVD28Apr84
: MENU-SELX    ." ? "   (S  ->  )
   BEGIN  UKEY  0                    \ Get key & set index
     PAD COUNT OVER +  SWAP          \ Scan allowed responses
     DO  OVER  I C@  =               \ -> char index flag
       ?LEAVE 3 +  3 +LOOP           \ -> char index
     PAD C@ 1- OVER <                \ Index past end? - yes
   WHILE DROP                        \ Drop index, check specials
     DUP  3  =  ABORT" Break"        \ Terminate on ^C
     DUP  ASCII  ?  =  IF (HELP)     ELSE
     1B =  IF  EXIT  ENDIF           \ exit MENU-SELX
     BEEP  ENDIF                     \ No BEEP after Help
   REPEAT  SWAP  EMIT  CR            \ -> index
   PAD + 2+ @ EXECUTE    ;           \ Get selected word addr


 .\ Define menus - HELP                                 FVD28Apr84
: HELP  CR CR  HELP# @ ." See help no:  " . CR  ;

' HELP IS  (HELP)
```

Figure Five
Execution Version of Menu Words

Michael Perry. It is readily adaptable to fig-FORTH, etc., though I don't know why anyone with an 8080/Z80 system would use anything other than Laxen and Perry's F83.

For those who are not aware, these two gentlemen implemented a meta-compiled version of Forth per the 83-Standard and put it in the public domain. It includes all kinds of bells and whistles, is one of the best Forth systems around, and the price is hard to beat. Most RCP/M computer bulletin boards around North America carry it as library SIGM154A.LBR and SIGM154B.LBR. Thanks Henry! Thanks, Michael!

```
 .\ Menus - using version 3                                FVD28Apr84
: MX8GX MX80 GRAFTRAX ; : MX10GX  MX100 GRAFTRAX  ;
: MX8NGX MX80 NOGTRAX ; : MX10NGX  MX100 NOGTRAX  ;

: MX80/100
  2 NEW-SEL  .TITLE   ." Set Printer Type"  CR CR
  .AX"  MX8NGX ASet MX80"
  .AX"   MX8GX BSet MX80 with Graftrax"
  .AX" MX10NGX CSet MX100"
  .AX"  MX10GX DSet MX100 with Graftrax"
  MENU-SELX  ;
\ Note that ," used will not skip leading blanks, hence only
\ one blank before menu text (applies to .A" and .AX")


 .\ Menus - Line spacing, Page size                        FVD28Apr84
: SPACING
  3 NEW-SEL  .TITLE   ." Set Line Spacing"  CR CR
  .AX"  LS-6 ASet 6 lines per inch"
  .AX"  LS-8 BSet 8 lines per inch"
  .AX" LS-72 CSet line spacing in 1/72 inch increments"
  (GTR) @  IF
    .AX"    LS-10 BSet 10.3 lines per inch"
    .AX"   LS-216 DSet line spacing in 1/216 inch increments"
    .AX" LS-216-1 ESet 1 line spacing in 1/216 inch increments"
  ENDIF  MENU-SELX  ;
: PAGE
  4 NEW-SEL  .TITLE   ." Set Page Size"  CR CR
  .AX"   PG-LN ASet page length in lines"
  .AX"   PG-IN BSet page length in inches"
  .AX" PG-WDTH CSet page width in characters"  MENU-SELX   ;
```

Figure Six
Sample of Execution Version of Menu Words

FORTH INTEREST GROUP MAIL ORDER FORM

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

MEMBERSHIP

IN THE FORTH INTEREST GROUP

107 - MEMBERSHIP in the FORTH INTEREST GROUP & Volume 7 of FORTH DIMENSIONS. No sales tax, handling fee or discount on membership. See the back page of this order form.

The Forth Interest Group is a worldwide non-profit member-supported organization with over 5,000 members and 80 chapters. FIG membership includes a subscription to the bi-monthly publication, FORTH Dimensions. FIG also offers its members publication discounts, group health and life insurance, an on-line data base, a job registry, a large selection of Forth literature, and many other services. Cost is $20.00 per year for USA, Canada & Mexico; all other countries may select surface ($27.00) or air ($33.00) delivery.

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

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

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

HOW TO USE THIS FORM

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

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

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

FORTH DIMENSIONS BACK VOLUMES

The six issues of the volume year (May - April) bound in a single text.

101 - Volume 1 FORTH Dimensions (1979/80) $15/16/18 ____

102 - Volume 2 FORTH Dimensions (1980/81) $15/16/18 ____

103 - Volume 3 FORTH Dimensions (1981/82) $15/16/18 ____

104 - Volume 4 FORTH Dimensions (1982/83) $15/16/18 ____

105 - Volume 5 FORTH Dimensions (1983/84) $15/16/18 ____

106 - Volume 6 FORTH Dimensions (1984/85) $15/16/18 ____

REFERENCE

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

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

316 - BIBLIOGRAPHY OF FORTH REFERENCES
2nd edition, Sept. 1984 .... $15/16/18 ____
An excellent source of references to articles about Forth throughout microcomputer literature. Over 1300 references.

ASSEMBLY LANGUAGE SOURCE CODE LISTINGS

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

513 - 1802/MARCH 81 .... $15/16/18 ____

514 - 6502/SEPT 80 .... $15/16/18 ____

515 - 6800/MAY 79 .... $15/16/18 ____

516 - 6809/JUNE 80 .... $15/16/18 ____

517 - 8080/SEPT 79 .... $15/16/18 ____

518 - 8086/88/MARCH 81 .... $15/16/18 ____

519 - 9900/MARCH 81 .... $15/16/18 ____

520 - ALPHA MICRO/SEPT 80............. $15/16/18 ____

521 - APPLE II/AUG 81.................... $15/16/18 ____

522 - ECLIPSE/OCT 82 .... $15/16/18 ____

523 - IBM-PC/MARCH 84................. $15/16/18 ____

524 - NOVA/MAY 81 .... $15/16/18 _____

525 - PACE/MAY 79 .... $15/16/18 _____

526 - PDP-11/JAN 80 .... $15/16/18 _____

527 - VAX/OCT 82 .... $15/16/18 _____

528 - Z80/SEPT 82 .... $15/16/18 _____

BOOKS ABOUT FORTH

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

205 - BEGINNING FORTH .... $17/18/21 _____
Paul Chirlian
Introductory text for 79-Standard.

215 - COMPLETE FORTH .... $16/17/20 _____
Alan Winfield
A comprehensive introduction including problems with answers. (Forth 79)

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

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

230 - FORTH FUNDAMENTALS, V. 2 .... $13/14/16 _____
Kevin McCabe
A glossary.

233 - FORTH TOOLS .... $19/21/23 _____
Gary Feierbach & Paul Thomas
The standard tools required to create and debug Forth-based applications.

237 - LEARNING FORTH .... $17/18/21 _____
Margaret A. Armstrong
Interactive text, introduction to the basic concepts of Forth. Includes section on how to teach children Forth.

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

245 - STARTING FORTH (soft cover) .... $20/21/22 _____
Leo Brodie (FORTH, Inc.)
A lively and highly readable introduction with exercises.

255 - THINKING FORTH (soft cover) .... $16/17/20 _____

260 - THINKING FORTH (hard cover) .... $23/25/28 _____
Leo Brodie
The sequel to "Starting Forth". An intermediate text on style and form.

265 - THREADED INTERPRETIVE LANGUAGES $23/25/28 _____
R.G. Loeliger
Step-by-step development of a non-standard Z-80 Forth.

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

FORML CONFERENCE PROCEEDINGS

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

310 - FORML PROCEEDINGS 1980 .... $25/28/35 _____
Technical papers on the Forth language and extensions.

311 - FORML PROCEEDINGS 1981 (2V) ...... $40/43/45 _____
Nucleus layer, interactive layer, extensible layer, metacompilation, system development, file systems, other languages, other operating systems, applications and abstracts without papers.

312 - FORML PROCEEDINGS 1982 .... $25/28/35 _____
Forth machine topics, implementation topics, vectored execution, system development, file systems and languages, applications.

313 - FORML PROCEEDINGS 1983 .... $25/28/35 _____
Forth in hardware, Forth implementations, future strategy, programming techniques, arithmetic & floating point, file systems, coding conventions, functional programming, applications.

314 - FORML PROCEEDINGS 1984 .... $25/28/35 _____
Expert systems in Forth, using Forth, philosophy, implementing Forth systems, new directions for Forth, interfacing Forth to operating systems, Forth systems techniques, adding local variables to Forth.

ROCHESTER PROCEEDINGS

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

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

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

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

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

THE JOURNAL OF FORTH APPLICATION & RESEARCH

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

401 - JOURNAL OF FORTH RESEARCH V.1 #1 $15/16/18 _____
Robotics.

402 - JOURNAL OF FORTH RESEARCH V.2 #1 $15/16/18 _____
Data Structures.

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

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

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

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

REPRINTS

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

421 - POPULAR COMPUTING 9/83 .... $5/6/7 _____
Special issue on various computer languages, with an in-depth article on Forth's history and evolution.

DR. DOBB'S

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

422 - DR. DOBB'S 9/82 .... $5/6/7 _____

423 - DR. DOBB'S 9/83 .... $5/6/7 _____

424 - DR. DOBB'S 9/84 .... $5/6/7 _____

HISTORICAL DOCUMENTS

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

502 - FIG-FORTH INSTALLATION MANUAL .. $15/16/18 _____
Glossary model editor - We recommend you purchase this manual when purchasing the source-code listings.

MISCELLANEOUS

601 - T-SHIRT SIZE ______________ $10/11/12 _____
(small, medium, large, extra large)

602 - POSTER (BYTE Cover) .... $15/16/18 _____

616 - HANDY REFERENCE CARD .... FREE _____

683 - FORTH-83 HANDY REFERENCE CARD ...... FREE _____

SELECTED PUBLICATIONS

The following publications are recent additions to the Forth Interest Group Order Form as selected by the FIG Publications Committee.

LEARNING FORTH, A Self-Teaching Guide by Margaret A. Armstrong. *Learning FORTH* takes you step-by-step through the various stages of programming, using a payroll program as an example.

In addition to developing a basic working knowledge of the language, *Learning FORTH* teaches good programming style and easy debugging techniques. There's also a section on how to teach children to use Forth.

1984 FORML CONFERENCE PROCEEDINGS
The sixth annual FORML Conference was held November 23-25, 1984 at the Asilomar Conference Center in Monterey, California, USA. Excellent articles in this volume include two papers on Expert Systems which won the acclaim of the participants. Many papers contain code you can use immediately, and some are philosophical papers which present challenges for the future of Forth.

FORTH-83 REFERENCE CARD
The Forth-83 Handy Reference Card is now available free of charge with any order. If you are not placing an order and wish to receive a card, please send a self-addressed, stamped envelope and we will be happy to send you one. Many thanks to Robert Berkey and Dave Kilbridge for compiling, designing and producing the card.

PUBLICATIONS SURVEY

If you would like to suggest any other publication for review by the FIG Publications committee for inclusion in the Forth Interest Group Order Form, please complete the information below and return to FIG.

Title: ______________________________

Author: ______________________________

Publisher: ______________________________

Comments: ______________________________

____

____

Your comments on any of the publications we currently carry are most welcome, please complete information below.

Title: ______________________________

Comments: ______________________________

____

____

FORTH INTEREST GROUP

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

Name ____________________

Company ____________________

Address ____________________

City ____________________

State/Prov. ____________ ZIP ____________

Country ____________________

Phone ____________________

OFFICE USE ONLY

By ________ Date ________ Type ________

Shipped By ________ Date ________

UPS Wt. ________ Amt. ________

USPS Wt. ________ Amt. ________

BO Date ________ Wt. ________ Amt. ________

By ________

ITEM #	TITLE	AUTHOR	QTY	UNIT PRICE	TOTAL
107	MEMBERSHIP				SEE BELOW

SUBTOTAL	
10% MEMBER DISCOUNT MEMBER # ________	
CA. RESIDENTS SALES TAX	
HANDLING FEE	$2.00
MEMBERSHIP FEE (NEW OR RENEWAL) $20/27/33	
TOTAL	

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

☐ VISA ☐ MASTERCARD

Card # ____________ Expir. Date ________

Signature ____________________

PAYMENT MUST ACCOMPANY ALL ORDERS

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

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

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

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

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

SALES TAX
California deliveries add 6%. San Francisco Bay Area add 7%.

7/85

Another Subroutine Technique

Donald Simard
Severn, Maryland

David Held's article and subsequent letter (*Forth Dimensions* V/3 and V/5) concerning code definitions callable by colon and code definitions, provided a very useful technique. After understanding his ideas, I came up with a variation which avoids some of the inherent execution-time overhead when his method is used from a colon definition.

Using Mr. Held's technique from a colon definition required storing the address of the subroutine that he wanted to call into the calling routine each time that word is executed. This overhead can be eliminated. Figure one illustrates what a subroutine definition will look like after being compiled using the defining words presented here.

One can see that the code field points to a **JSR** to the parameter field address. The code that is then to be executed must be a machine language subroutine and must return accordingly. When the word is called from a colon definition, the inner interpreter will be directed through the code field to the **JSR PFA**. When the machine language subroutine in the parameter field returns a **JMP** then **NEXT** is executed, which is the normal exit for **CODE** definitions. If the subroutine is to be used from a code definition, first it is necessary to use ' to find the **PFA**, then **JSR** to this address.

The execution overhead associated with this technique is just an extra **JSR** and **RTS** from a normal colon definition; there is no overhead from code definitions. The extra memory required is the four bytes for the **JSR** and **JMP** in the definition.

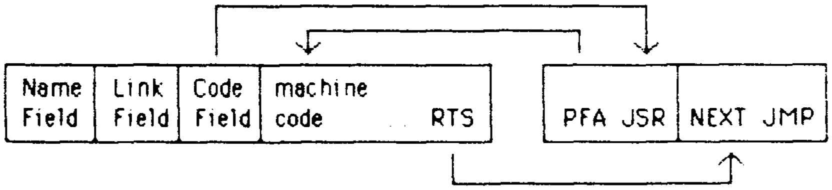

Figure One

```
: SUB
      [COMPILE] ASSEMBLER
      CREATE SMUDGE
      LATEST PFA CFA
;

: END-SUB
      ASSEMBLER
      HERE OVER !
      [COMPILE] ASSEMBLER
      2+ JSR, NEXT JMP,
      END-CODE
;

EXAMPLE:
          SUB TEST
                41 # LDA,
                PRINT JSR,
                RTS,
           END-SUB

CALLED FROM COLON:
          : PRINT.1
                TEST
          ;

CALLED FROM CODE:
          CODE PRINT.1
                ' TEST JSR,
                NEXT JMP,
           END-CODE
```

The Mirth Dimension

```
: PROJECT_DIRECTOR    EXPECT UPDATE VERSION  ,  WARNING IF SILENT
                      TOGGLE WRITE_CHECK AND EXPECT COLD HANDSHAKE
                      ELSE COMPILE AGAIN AND B SILENT  ;

: FORTH_PROGRAMMER    TRAVERSE COLD SILENT OUTER SPACE AND CREATE
                      RANDOM DEFINITIONS  ;

: FORTH_SOURCE_CODE   EMIT RANDOM WORD AND REPLACE ERROR UNTIL
                      SPACE = MAX LIMIT OR TIME OVER  ;
```

— Scott Heiner &
Steve Gledhill

Hacker's LOCKER

Cecil McGregor
Santa Clara, California

Many times, when debugging Forth code, it is convenient to use small "throw away" parts that will not be used in the later, bug-free version. This could include a few lines of output that would be handy to keep on the display, or portions of a dump to compare with something else. It is not always desirable to edit this onto a screen and reload from that for changes, nor is it always convenient to print it as hard copy. These small gems then scroll off the top of the screen. Or perhaps you are looking at the stack and would like to keep it around while you scrounge about analyzing a bug. A few lines of typing and it, too, goes off the top of the screen.

If you have a terminal that permits line lock (such as a TeleVideo 950 and many others) you can simulate a window that will lock the contents of the line until you release it. Scrolling will then ignore these locked lines. While the appearance is rather odd at first, the usefulness is great.

To implement this, you must look in your terminal manual for the codes that will lock a line, unlock a line or screen, move up and down a line. As can be seen in the accompanying code, the word **LINE-LOCK** locks; and the word **SCR-UNLOCK** unlocks the entire screen. The constants **UP-ARROW** and **DN-ARROW** move the cursor up and down a line, respectively. Each of these is terminal dependent and must be tailored to your terminal.

The word **LOCKER** gets a keystroke, and moves up or down a line if the up and down arrows are input. **UP-ARROW** and **DN-ARROW** allow positioning of the cursor onto the line to be locked. Striking an "L" will cause the current line to be locked; a "U" will unlock the entire screen; and any other key will exit **LOCKER**.

This screen will compile under fig-FORTH and Laboratory Microsystems' 83-Forth, and it should be easily transportable to almost any standard Forth. If your terminal has line-lock capabilities, you will like this utility.

```
Screen # 110
 \ LOCKER - lock/unlock lines on page                        CHM11JUL84
   HEX
 : LINE-LOCK  ( --- ) ( TV950 control to lock a line )
     1B EMIT   21 EMIT   31 EMIT ;
 : SCR-UNLOCK ( --- ) ( TV950 control to unlock entire screen )
     1B EMIT   21 EMIT   32 EMIT ;
 0B CONSTANT UP-ARROW     16 CONSTANT DN-ARROW
 : LOCKER ( --- ) ( selectively lock lines on crt )
    BEGIN KEY
      DUP UP-ARROW = IF       EMIT              ELSE  ( move up    )
      DUP DN-ARROW = IF       EMIT              ELSE  ( move down  )
      DUP 4C ( L ) = IF DROP LINE-LOCK          ELSE  ( lock       )
      DUP 55 ( U ) = IF DROP SCR-UNLOCK         ELSE  ( scr unlock)
                        DROP  QUIT          ( any other key exits. )
       THEN THEN THEN THEN
    AGAIN ; DECIMAL
```

Mass Transit Forth

From June onwards, some bus stops in the resort and retirement town of Weston-super-Mare, Somerset, U.K., will never be at a loss for words. Waiting passengers will hear a quiet beeping and, if they then press a button at the source of the sound, a personable female voice will announce the time of day, what buses are using that stop and when they are due.

When a bus is running off schedule, inductive loop sensors buried in the road interrogate circuitry in the bus cab to identify it. The microprocessor controller in the speech synthesizer scans the sensor, detects the oncoming bus and a voice will announce something like, "The bus now approaching is a No. 10." Meanwhile, the controller figures out when the next one is due.

London-based Triangle Digital Services Ltd., a one-man outfit set up by Peter Rush, developed the prototype hardware for the system. Formerly a product manager with General Instrument Microelectronics Ltd., Rush set up shop to provide a systems service to potential users of voice systems.

He has developed a custom speech chip using a digital logic array. This is mounted on a self-contained Eurocard-sized board. To go with it, there is a Forth-language microcomputer board. Rush says he chose Forth because its high-level-language capability speeds development work.

Using the speech-synthesizer board and the microcomputer board (which serves as its own development system), Rush can quickly meet a customer's requirements, even tailoring the voice to the applications. For Britain's National Physical Laboratory, for example, Triangle has developed a handheld keypad with sixteen-digit alphanumeric display. It is used to check readings as they are entered into the laboratory computer. With the synthesizer speaking in a brisk manner with truncated enunciation, it can keep up with the user as he keys in a long numeric string.

The terminal can be used both for entering data and instructions and for receiving prompts from the laboratory computer. For example, it could give the operator a cue about what reading to make next or indicate that a reading is out of range. The terminal communicates with the laboratory computer systems through an RS-232-C interface at 9,600 bits per second or through an IEEE-488 port.

Triangle has now acquired rights to the terminal from NPL and is also negotiating with the Department of Transport for the rights to the passenger advisory service, which clearly has potential applications in road, rail and airplane terminals, just for starters.

For those companies with sufficient expertise to assemble their own systems, Triangle also sells its various board products directly. One is a speech-synthesizer board that can be operated directly from relay contacts, asynchronous links over twisted pairs or parallel binary-coded decimal inputs.

Forth Talk

Triangle has also developed a Forth development board based on a Hitachi 6303 — a complementary-MOS variant of a Motorola part — and running Triangle's own Forth system. The board is a useful tool for real-time applications, since multi-tasking programs can be written in a high-level language.

Moreover, users can extend the language as they go along, building their own expertise into their systems. Rush has used this feature to develop progressively more sophisticated speech-control algorithms. The result has been speech with very acceptable quality.

The Department of Transport system shows off the microcomputer's multitasking capability. First, it keeps a calendar that tracks the minute, hour, day and month. It responds to changes in summer and winter schedules and switches to the appropriate timetable.

Second, it scans the inductive loop sensors and decodes the detected signal. Next, it controls the speech-synthesizer chip's vocabulary and adjusts the volume to compensate for ambient noise levels. "The lady shouts when a lorry goes by," says Rush. There is also a switched-capacitor-filter board with a twelve-decibel-per-octave cut-off.

—Kevin Smith

Part Two

Forth Spreadsheet

Craig A. Lindley
Manitou Springs, Colorado

1
```
\ spreadsheet - loader block                          cl 09/21/84

dark .( Spreadsheet Compiling)      warning off

1  45 +thru

\s
load the spreadsheet into the forth dictionary
if you wish to save the spreadsheet, do so as follows:

          ' spreadsheet is boot    <cr>
          save-system filename.com <cr>

this will save the entire forth system along with the spread-
sheet as an executable file named filename. When the file
is executed, your spreadsheet will execute automatically.
```

2
```
\ spreadsheet - case statement                        cl 06/28/84

: ?comp   state @ not abort" Compilation only" ;
: ?pairs  <>        abort" Bad CASE statement"  ;

: case    ?comp csp @ !csp 4 ; immediate
: of      4 ?pairs
          compile over compile = compile ?branch
          here 0 , compile drop 5 ; immediate
: endof   5 ?pairs compile branch here 0 ,
          swap   >resolve 4 ; immediate
: endcase 4 ?pairs compile drop
          begin sp@ csp @ <>
          while >resolve repeat
          csp ! ; immediate
```

3
```
\ spreadsheet - constant & variable declarations     cl 06/24/84


26 constant row_max         \ # of spreadsheet rows
26 constant col_max         \ # of spreadsheet columns
12 constant col_name_len    \ max length of column name
17 constant row_name_len    \ max length of row    name
21 constant col_org         \ column origin of data on display
 3 constant row_org         \ row    origin of data on display
 6 constant bytes/cell      \ # of bytes per cell
```

4
```
\ spreadsheet - constant & variable declarations     cl 07/04/84

variable mode_flag               \ auto calculate flag
variable order_flag              \ calculation order flag
variable format_flag             \ number output format flag
variable cur_col                 \ top left display column #
variable cur_row                 \ top left display row    #
variable col_disp                \ column displacement from
                                 \ cur_col on display
variable row_disp                \ row    displacement from
                                 \ cur_row on display
variable dict_mark               \ beginning of formula area
variable op_stack 44 allot       \ operator stack for algebraic
                                 \ equation compilation
```

5
```
\ spreadsheet - high level array definitions          cl 06/31/84

\ create 2D array depth bytes deep
: array        \ ( #rows #cols depth -- )        compile time
               \ ( row#  col# -- element addr ) run     time
  create 2dup swap c, c, * * dup here
  swap erase allot
  does> dup c@ 3 roll * 2 roll + over
       1+ c@ * + 2+ ;

\ create 1D string array depth characters deep
: $array       \ ( #rows depth -- )              compile time
               \ ( row# -- string addr )         run     time
  create dup c, * dup here swap blank allot
  does>  dup c@ rot * + 1+ ;
```

6
```
\ spreadsheet - array definitions                     cl 06/28/84

\ define a 2D array for spreadsheet data structure
\ each cell contains 6 bytes
\ 2 for formula execution address (if any)
\ 4 for double number value storage

row_max col_max bytes/cell array cells

\ define a string array for holding the row names
row_max row_name_len $array row_names

\ define a string array for holding the column names
col_max col_name_len $array col_names
```

7

```
\ spreadsheet - misc word definitions              cl 07/04/84
: IBM_key                      \ input special IBM keys
  key ?dup 0=                  \ ( -- c )
  if key 128 + then ;

: d#in                         \ input double # from keyboard
  pad 1+ 20 2dup blank expect  \ ( -- d)
  span @ pad c! pad number ;

: #in d#in drop ;              \ input single number
                               \ ( -- n)
```

8

```
\ spreadsheet - misc word definitions              cl 07/02/84
: pos1 0 21 2dup at 0 blot at ; \ position cursor on cmd line
: pos2 0 22 2dup at 0 blot at ; \ position cursor on cmd line

: y/n                          \ ( -- T if yes F if no )
  pos1 ." Are You Sure ?: "    \ display msg
  key upc ascii Y = ;          \ return flag

: mark_cell                    \ mark cell on display
  2dup at ascii < emit swap 11 \ (row# col# -- )
  + swap at ascii > emit ;     \ mark cell like "<       >"

: unmark_cell                  \ unmark cell of display
  2dup at space swap 11 + swap \ (row# col# -- )
  at space ;                   \ remove <  > marks from display
```

9

```
\ spreadsheet - misc word definitions              cl 07/03/84
: cell_ptr                     \ return address of cell pointed
  cur_row @ row_disp @ +       \ at by <   > display marker
  cur_col @ col_disp @ +       \ ( -- cell addr )
  cells ;

: cal_cell_disp_loc            \ calculate location on display
  col_disp @ 13 * col_org +    \ of cell display markers
  row_disp @      row_org + ;  \ ( -- col row )

: place_cell_marker            \ place cell marker around cell
  cal_cell_disp_loc mark_cell ;

: erase_cell_marker            \ erase cell marker around cell
  cal_cell_disp_loc unmark_cell ;
```

10

```
\ spreadsheet - misc word definitions              cl 07/04/84
: (fd.) tuck dabs              \ dollar/cents formating word
  <# # # ascii . hold #s rot   \ formats double # on TOS
   sign ascii $ hold #> ;      \ prints leading $

: fd.r >r (fd.) r>             \ format d# in dollars/cents in
  over - spaces type ;         \ right justified field width w
                               \ (d w -- )
: format#                      \ format double number in one of
  format_flag @                \ two formats
  if   10 fd.r                 \ as dollars/cents
  else 10  d.r                 \ normal number
  then ;
```

11

```
\ spreadsheet - display word definitions           cl 07/04/84

: dis_data                     \ display all cell data
  cur_col @ dup 4 + swap       \ for 4  screen columns
  do i col_max = ?leave        \ if past last possible column
   cur_row @ dup 15 + swap     \ for 15 screen rows
   do i row_max = ?leave       \ if past last possible row
    j cur_col @ - 13 * 22 +    \ calculate display col position
    i cur_row @ -       3 + at \ calculate display row position
    i j cells 2+ 2@ format#    \ get data and display it
   loop                        \ loop for all data displayed
  loop ;
```

12

```
\ spreadsheet - display word definitions           cl 07/02/84
\ display spreadsheet boarders on the screen

: dis_boarder                  \ display spreadsheet boarders
  18 3                         \ ( -- )
  do 20 i at 4 0
     do ascii | emit 12 spaces
     loop ascii | emit
  loop
  80 0
  do i  2 at ascii - emit loop
  80 0
  do i 18 at ascii - emit loop ;
```

13

```
\ spreadsheet - display word definitions          cl 07/04/84

\ display spreadsheet menu of options on right side of display

: dis_menu
  74  3 at ." Menu:"
  74  4  at ." C)ol"   74 5  at ." A)gain"
  74  6 at ." D)ata"   74 7  at ." E)qu."
  74  8 at ." F)orm."  74 9  at ." G)oto"
  74 10 at ." M)ode"   74 11 at ." N)ew"
  74 12 at ." O)rder"  74 13 at ." P)ref"
  74 14 at ." Q)uit"   74 15 at ." R)ow" ;
```

14

```
\ spreadsheet - display word definitions          cl 07/04/84

: dis_row_labels                \ label the rows on display
  cur_row @ dup 15 + swap       \ label from cur_row
  do i row_max = ?leave         \ for 15 lines or until row_max
     18 i cur_row @ - 3 + at    \ format in decimal #s
     i 2 .r
  loop ;

: dis_row_names                 \ display row names from array
  cur_row @ dup 15 + swap       \ only show names that fit on
  do i row_max = ?leave         \ display
     0 i cur_row @ - 3 + at     \ place cursor at location
       i row_names row_name_len \ type required chars
         type
  loop ;
```

15

```
\ spreadsheet - display word definitions          cl 07/03/84
: dis_col_labels                \ label the columns on display
  cur_col @ dup 4 + swap        \ label from cur_col
  do i col_max = ?leave         \ for 4 cols. or until col_max
     i cur_col @ - 13 * 27 +    \ format in alphabetic chars
     2 at i ascii A + emit      \ A thru Z
  loop ;

: dis_col_names                 \ display column names
  7 1 at ." Title"              \ only show names that fit on
  cur_col @ dup 4 + swap        \ display
  do i col_max = ?leave
     i cur_col @ - 13 * 21 +    \ place cursor at location
     1 at i col_names col_name_len
       type                     \ type required # of chars
  loop ;
```

16

```
\ spreadsheet - display word defintions           cl 06/31/84

: dis_status                    \ display spread sheet status
  48 19 at ." Row: "            \ display current row/col
  cur_row @ row_disp @ + .
  60 19 at ." Column: "
  cur_col @ col_disp @ + ascii A + emit
  47 20 at ." Mode: " mode_flag @       \ display calc. mode
  if ." Auto  " else ." Normal" then
  61 20 at ." Order: " order_flag @     \ display calc. order
  if ." C/R" else ." R/C" then
  pos2
  pos1 ." Command: "            \ output command prompt
  place_cell_marker ;           \ place cell marker on display
```

17

```
\ spreadsheet - display word definitions          cl 06/31/84

: dis_row_change                \ display info that changes with
  dis_row_names                 \ a row change
  dis_row_labels                \ row names, labels and data
  dis_data ;

: dis_col_change                \ display info that changes with
  dis_col_names                 \ a col change
  dis_col_labels                \ col names, labels and data
  dis_data ;
```

18

```
\ spreadsheet - display word definitions          cl 07/04/84
: dis_screen                    \ display spreadsheet screen
  dark 31 0 at
  ." Forth Spreadsheet"         \ display title
  dis_boarder                   \ draw boarders
  dis_menu                      \ display operation menu
  dis_col_labels                \ label columns (A-Z)
  dis_col_names                 \ display column names
  dis_row_labels                \ label rows    (0-25)
  dis_row_names                 \ display row    names
  dis_data                      \ display appropriate data
                                \ for data window being
                                \ displayed
  0 row_disp ! 0 col_disp !     \ set mark at origin
  dis_status ;                  \ display status
```

19

```
\ spreadsheet - cell calculation words           cl 06/31/84

: calculate                    \ calc formula of cell if it has
  @ ?dup if execute then ;     \ one ( cell addr -- )

: calc_c/r                     \ calc columns then rows
  row_max 0
  do col_max 0
     do j i cells calculate loop  \ get formula and execute it
  loop ;

: calc_r/c                     \ cal rows then columns
  col_max 0
  do row_max 0
     do i j cells calculate loop  \ get formula and execute it
  loop ;
```

20

```
\ spreadsheet - cell calculation words           cl 07/04/84

: calc_cells                   \ determine which to calc first
  order_flag @                 \ by state of order_flag
  if calc_c/r                  \ if 1 calc cols then rows
  else calc_r/c                \ if 0 calc rows then cols
  then ;

: order                        \ prompt user for calc order
  pos1 ." Specify calculation order"
  pos2 ." Row/Col(0) or Col/Row(1): "
  key ascii 1 =                \ get response and set flag
  if true                      \ accordingly
  else false
  then order_flag ! ;
```

21

```
\ spreadsheet - cell marker positioning words    cl 07/02/84

: left_arrow                   \ move cell marker left 1 cell
  col_disp @ 0=                \ cell mark at left of display ?
  if cur_col @ 0 <>            \ if so is at at first column  ?
     if -1 cur_col +!          \ move display column left once
          dis_col_change       \ scroll display left
     then                      \ if at first column ignor
  else                         \ cell mark not at left column
     erase_cell_marker         \ erase current mark
     -1 col_disp +!            \ move left without scroll
  then                         \ of display
  place_cell_marker ;          \ draw new cell marker
```

22

```
\ spreadsheet - cell marker positioning words    cl 07/02/84

: right_arrow                  \ move cell marker right 1 cell
  col_disp @ 3 =               \ marker in right most cell ?
  if cur_col @  4 + col_max <> \ if so is it the last cell (Z)
     if 1 cur_col +!           \ move display column right one
        dis_col_change         \ scroll right
     then                      \ if mark at column Z ignor
  else                         \ if mark not at right most
     erase_cell_marker         \ column of display move it
     1 col_disp +!             \ right one cell without
  then                         \ scroll
  place_cell_marker ;          \ draw new cell marker
```

23

```
\ spreadsheet - cell marker positioning words    cl 07/02/84

: up_arrow                     \ move cell marker up 1 cell
  row_disp @ 0=                \ is cell at top display pos. ?
  if cur_row @ 0 <>            \ if so are we at the top of
     if -1 cur_row +!          \ the spreadsheet ?
                               \ if not move up a
        dis_row_change         \ cell and scroll upward
     then                      \ if already at top ignor
  else                         \ if mark not at top of display
     erase_cell_marker         \ erase mark
     -1 row_disp +!            \ move up one cell
  then
  place_cell_marker ;          \ draw new cell marker
```

24

```
\ spreadsheet - cell marker positioning words    cl 07/02/84

: down_arrow                   \ move cell marker down 1 cell
  row_disp @ 14 =              \ are we at bottom of display ?
  if cur_row @ 15 + row_max <> \ if so are we on last row    ?
     if 1 cur_row +!           \ if not move down one cell
        dis_row_change         \ scroll downward
     then                      \ if at last row ignor
  else                         \ if not at bottom of display
     erase_cell_marker         \ erase cell mark
     1 row_disp +!             \ move down one cell
  then
  place_cell_marker ;          \ draw new cell marker
```

```
25
\ spreadsheet - cell marker positioning words       cl 07/04/84

: first_col 0 cur_col !        \ go to column A immediately
  dis_col_change ;             \ cur_col to 0 and scroll

: last_col                     \ go to column W immediately
  col_max 4 - cur_col !        \ cur_col to W and scroll
  dis_col_change ;

: top_row                      \ go to row 0 immediately
  0 cur_row !                  \ cur_row to 0 and scroll
  dis_row_change ;

: bottom_row                   \ go to column 11 immediately
  row_max 15 - cur_row !       \ cur_row to 11 and scroll
  dis_row_change ;
```

```
26
\ spreadsheet - cell marker positioning words       cl 07/04/84

: left_4_cols                  \ move marker left  4 columns
  4 0 do left_arrow  loop ;    \ at a time

: right_4_cols                 \ move marker right 4 columns
  4 0 do right_arrow loop ;    \ at a time
```

```
27
\ spreadsheet - algebraic functions                 cl 07/04/84
vocabulary algebra             algebra also definitions

\ col_id function assigns n to id at compile time (n -- )
\ expects row # on TOS at run time
\ subsequent usage of id fetches double value of cell to stack

: col_id                       \ column_id high level defining
  create ,                     \ word. Creates col ids A-Z
  does> @ cells 2+             \ expect a # on the TOS and
  2@ ;                         \ pushes the cell value onto
                               \ the parameter stack
```

```
28
\ spreadsheet - algebraic functions                          cl 07/04/84

: assign_id col_max 0          \ loop used to assign values to
  do i col_id loop ;           \ the alphabetic columns

assign_id     A B C D E F G H I J K L M N O P
                 Q R S T U V W X Y Z

\s

for example: 1 A returns the double int value of   cell 1 A

Column ids A-Z return values of 0-25 respectively
```

```
29
\ spreadsheet - algebraic functions                          cl 06/31/84
: opp@                              \ return oprand stack position
  op_stack dup @ + ;                \ 1st location is stack ptr
                                    \ ( -- addr )
: >op 4 op_stack +! opp@ 2! ;       \ store cfa and precedence
                                    \ top of oprand stack
                                    \ (cfa prec -- )
: op> opp@ 2@ -4 op_stack +!        \ pop cfa and prec off oprand
  drop , ;                          \ stack and compile into dict.
: prec? opp@ @ ;                    \ return precedence from top
                                    \ top of oprand stack
                                    \ ( -- prec)
: ]a begin prec?                    \ end algebraic compilation
  while op> repeat                  \ pop remaining oprands off stk
  forth ; immediate                 \ and compile then select forth
                                    \ vocabulary again
```

```
30
\ spreadsheet - algebraic functions                          cl 07/04/84
\ create high level definition that performs algebraic
\ compilation. See text for details of operation

: infix  '                          \ create new algebraic operator
  create swap , , immediate         \ compile cfa of forth operator
  does> 2@                          \ and assigned precedence
  begin dup prec? > not             \ at compile time execute if
  while >r >r op> r> r>             \ prec is lower then oprand on
  repeat >op ;                      \ top of oprand stack

: d* dup rot * rot rot um* rot + ; \ double multiplication
: d/ swap over /mod >r swap         \ double division
  um/mod swap drop r> ;
: dmod d/ drop 0 ;                  \ double modulus
```

31

```
\ spreadsheet - algebraic functions                    cl 07/04/84

7 infix d* *      7 infix d/ /    \ create new algebraic operators
6 infix d+ +      6 infix d- -    \ with assigned precedence
5 infix dmod mod

: )missing                        \ missing ) message
  1 abort" Missing )" ;           \ if missing then abort

: (                               \ left paren
  ['] )missing 1 >op ;            \ prec=1 cfa=)missing message
  immediate                       \ push on oprand stack
```

32

```
\ spreadsheet - algebraic functions                    cl 06/31/84

: ) [ forth ]                     \ right paren
    begin 1 prec? <               \ causes all items on oprand
    while op> repeat              \ stack to be compiled until
    1 prec? =                     \ left paren found
    if -4 op_stack +!             \ left paren should have prec.
    else 1 abort" Missing ("      \ of 1 else error msg output
    then ; immediate

forth definitions

: a[ 0 op_stack ! algebra ;       \ start algebraic compilation
  immediate                       \ reset oprand stack and
                                  \ select algebra vocabulary
```

33

```
\ spreadsheet - input words                            cl 07/03/84
: input_row_names                 \ input row names
  pos1 ." Input Row Names"        \ display command prompt
  pos2 ." Starting with Row: "
  #in cur_row !                   \ get starting row # store
  row_max cur_row @               \ in cur_row. From there till
  do pos2                         \ last row input names
     ." Row " i 2 .r ." : "       \ display row #
     i row_names row_name_len     \ get address in row_name array
     2dup blank expect            \ clear it and then input
     span @ 0= ?leave             \ if just <CR> then exit loop
     i 5 mod 0=                   \ every 5 names scroll display
     if i cur_row !               \ new cur_row change display
        dis_row_change            \ else only change the
     else dis_row_names then      \ display row names
  loop ;
```

34

```
\ spreadsheet - input words                            cl 07/03/84
: input_col_names                 \ input column names
  pos1 ." Input Col Names"        \ display command prompt
  pos2 ." Starting with Col: "
  key upc ascii A - cur_col !     \ get starting col # store
  col_max cur_col @               \ in cur_col. From there till
  do pos2                         \ last col input names
   ." Col " i ascii A + emit ." : "   \ display col #
   i col_names col_name_len       \ get address in col_name array
   2dup blank expect              \ clear it and then input
   span @ 0= ?leave               \ if just <CR> then exit loop
   i 4 mod 0=                     \ every 4 names scroll display
   if i cur_col !                 \ new cur_col change display
      dis_col_change              \ else only change the
   else dis_col_names then        \ display col names
  loop ;
```

35

```
\ spreadsheet - input words                            cl 07/04/84

: get#                            \ get double # scale if needed
  d#in                            \ get double int from operator
  format_flag @                   \ dollars/cents format ?
  if dpl @ 3 min                  \ if so scale accordingly
   case
    -1 of 100 d* endof            \ no DP value * 100
     0 of 100 d* endof            \  0 DP value * 100
     1 of  10 d* endof            \  1 DP value *  10
     2 of  noop  endof            \  2 DPs no scaling
     3 of  10 d/ endof            \  3 DPs value / 10
   endcase
  then ;                          \ if normal format
                                  \ no scaling performed
```

36

```
\ spreadsheet - input words                            cl 07/04/84

: input_cell_data                 \ input data to cell
  pos1 ." Input Cell Data"        \ prompt for data entry
  pos2 ." Data: " get#            \ get data
  cell_ptr 2+ 2!                  \ and store it
  mode_flag @                     \ get mode flag
  if pos2 ." Calculating"         \ if auto calculate mode
     calc_cells                   \ selected then calculate
  then                            \ all cell data
  dis_data ;                      \ show the new data
```

```
37
\ spreadsheet - input words                          cl 09/21/84
: input_equ                       \ input equation into dict
  pos1 ." Input Cell Equation"    \ prompt for equation
  pos2 ." Equation: "
  tib 127 blank                   \ clear tib
  " : formula a[ "                \ preamble to move to tib
  tib swap cmove                  \ move it to tib
  tib 13 + dup 127 expect         \ get equation to tib
  span @ +                        \ pt at end of input
  "  ]a [ cell_ptr 2+ ] literal 2! ; last @ name> cell_ptr !"
  -rot swap cmove                 \ move to tib also
  span @ 70 + #tib !              \ make forth think it all
  blk off >in off                 \ came from the keys
  algebra                         \ select algebra vocabulary
  interpret                       \ compile equation into dict
  forth ;                         \ back to forth  vocabulary
```

```
38
\ spreadsheet - high level commands                  cl 07/04/84
: quit_calc                    \ exit spread sheet
  y/n abort" BYE" ;            \ ask again if YES then quit

: new y/n                      \ clear existing spreadsheet
   if 0 0 cells                \ ask again if yes clear it
    row_max col_max bytes/cell
    * * erase                  \ erase cells array
    0 row_names
    row_max row_name_len * erase \ erase row_name array
    0 col_names
    col_max col_name_len * erase \ erase col_name array
    dict_mark perform          \ erase all formulas
    0 row_disp ! 0 col_disp !  \ set marker to origin
    dis_screen                 \ display cleared screen
   then ;
```

```
39
\ spreadsheet - high level commands                  cl 07/03/84
: mode                         \ set auto calculation mode
  pos1
  ." Set auto calculation mode"
  pos2
  ." Normal=0 or Auto=1: "     \ prompt operator
  key ascii 1 =                \ get response
  if   true                    \ set mode_flag
  else false                   \ accordingly
  then
  mode_flag ! ;

: perform_calc                 \ force calculations
  calc_cells                   \ execute formulas
  dis_data ;                   \ show result
```

```
40
\ spreadsheet - high level commands                  cl 07/04/84

: format                        \ select # format
  pos1 ." Select input number format"
  pos2 ." Normal=0 or Dollars/Cents=1: "
  key ascii 1 =                 \ get operator response
  if   true                     \ if $ then true  to format_flag
  else false                    \ otherwise false to format_flag
  then
  format_flag !
  dis_data ;                    \ show data in new format
```

```
41
\ spreadsheet - high level commands                  cl 07/04/84

: again_repl                    \ replicate column data
  cell_ptr 2+ 2@                \ bring cell data to TOS
  pos1 ." Column replicate cell data"
  pos2 ." Number of columns: "
  #in ?dup                      \ get # of columns
  if 0                          \ if answer <> 0
   do right_arrow               \ move right
      2dup cell_ptr 2+ 2!       \ and store data
   loop
   2drop                        \ clean up the stack
   dis_data                     \ and display the new data
  then ;                        \ else ignor if col=0
```

```
42
\ spreadsheet - high level commands                  cl 07/04/84

: go_to                         \ goto specified row/col
  pos1 ." Row(0-25): "          \ prompt for row #
  #in dup 0 row_max within      \ check for proper range
  if cur_row !                  \ if ok store it
   pos2 ." Column(A-W): "       \ prompt for col letter A-W
   key upc ascii A - dup        \ get it and check its range
   0  col_max 3 -  within       \ if ok goto data window
   if cur_col !                 \ store col #
    0 col_disp ! 0 row_disp !   \ place cell marker at origin
    dis_screen                  \ show new screen
   else drop                    \ drop #s if out of range
   then
  else drop
  then ;
```

```
    43                                                              0
\ spread sheet - operator input processing          cl 07/04/84
: command_in case
   ascii A of again_repl        endof \ replicate cell data
   ascii C of input_col_names   endof \ input column names
   ascii D of input_cell_data   endof \ input cell data
   ascii E of input_equ         endof \ input cell equation
   ascii F of format            endof \ input # display format
   ascii G of go_to             endof \ goto  cell
   ascii M of mode              endof \ set calc mode
   ascii N of new               endof \ clear spreadsheet
   ascii O of order             endof \ set calc order
   ascii P of perform_calc      endof \ force calculations
   ascii Q of quit_calc         endof \ quit  spreadsheet
   ascii R of input_row_names   endof \ input row names
   beep                               \ if none of the above
  endcase ;

    44                                                              0
\ spread sheet                                      cl 07/04/84
: control_in
  case
    199 of top_row      endof     \ goto top    row   of sheet
    200 of up_arrow     endof     \ up    one cell
    201 of left_4_cols  endof     \ go left 4 columns
    203 of left_arrow   endof     \ left  one cell
    205 of right_arrow  endof     \ right one cell
    207 of bottom_row   endof     \ goto bottom row   of sheet
    208 of down_arrow   endof     \ down  one cell
    209 of right_4_cols endof     \ go right 4 columns
    243 of first_col    endof     \ goto first column of sheet
    244 of last_col     endof     \ goto last  column of sheet
    beep                          \ if none of above
  endcase ;


    45                                                              0
\ spreadsheet - main program                        cl 09/21/84

: spreadsheet                 \ main program word
  dis_screen                  \ show screen
  begin
    IBM_key upc               \ get key convert to UC
    dup 198 >                 \ control or command key ?
    if   control_in           \ control key input
    else command_in           \ command key input
    then
    dis_status                \ show new status
  again ;                     \ do forever

mark no_formulas              \ dict_mark has cfa of
' no_formulas dict_mark !     \ word to delete formulas
warning on                    \ warning msgs back on
```

Availability of Spreadsheet Source Code and F83

To save some rather tough typing, the source code for this program is available postage paid from the author for $25 (6 Sutherland Place, Manitou Springs, Colorado 80829). It is, however, only available on 5.25″ disk format for MS-DOS or PC-DOS. The public-domain F83 program should be available from No Visible Support Software, Mike Perry, 1125 Bancroft Way, Berkeley, California 94702.

This is an incredible implementation of Forth-83. Everyone even remotely interested in Forth should buy a copy of it. The entire Forth community should give thanks to Laxen and Perry for donating such an excellent piece of software to the public domain.

The Mirth Dimension

2SWAP DROPY

With apologies to Lewis Carroll.

'Twas SYSOUT and the SFILL toves
  Did SRW in the wabe;
All FOUTPUT were the borogoves,
  And the BOOT COLD outgrabe.

"Beware the 2SWAP DROP my son!
  The ROLLS that bite, the DOs that catch!
Beware the DLIT LOOP and shun
  The NORETCOND Bandersnatch!"

He took his OUTER PICK in hand!
  Long time the PCOND PORT he sought.
So rested he by the IF ELSE tree,
  And FLUSH PMODE in thought.

And as in PIXBLT thought he stood,
  The 2SWAP DROP, with ROTs of flame,
Came TONED through the /MOD wood
  And <BUILDS DOES> as it came!

One, two! One, two! And through and through
  The OUTER PICK went snicker-snack!
He left it WARNING and with its head
  He went CPATH back.

"And hast thou slain the 2SWAP DROP?
  Come D1+ my QUSER boy!
A TYPED day! Callooh callay!"
  He RANDOM in his joy.

'Twas SYSOUT and the SFILL toves
  Did SRW in the wabe;
All FOUTPUT were the borogoves,
  And the BOOT COLD outgrabe.

— Wayne Cox

Rochester Forth Conference 1985

Forth programmers, project managers, vendors and evaluators gathered in Rochester, New York in June at the annual Forth conference held there. The directors and staff of the Institute for Applied Forth Research assembled a broad spectrum of presentors who gave us a clear profile of the expertise being brought to bear in today's Forth community. From applications on the space shuttle and automation of an entire airport, to object-oriented programming, to putting Forth in hardware, the papers were intelligent, sometimes witty, and occasionally mind boggling.

Elizabeth Rather of Forth Inc. spoke on the subject of developing and implementing a large-scale application. Titled "Fifteen Programmers, 400 Computers, 36,000 Sensors and Forth," the talk described work done to automate and fully integrate an international airport in the Middle East. Major applications such as safety, security, climate control, personnel, power distribution, runway lighting, etc., all reside in one large system, with largely common subsystems providing facilities to each. Forth lends itself well to this method of using modular common factors to serve in diverse areas.

Rather described the months of strategy, planning and testing required by the project. Much of the work was done in Alabama, where the team worked in two-weeks-on, one-week-off shifts. A huge hangar-like room was used to house a crowded maze of cables and interconnected machines — each programmable from any of the others — that represented only about a third of the actual site equipment.

After describing the size and complexity of this task, the speaker went on to offer her observations about the requirements for success with large projects: knowledgeable management, and direction; professional skill in all parts of the working team; good communication between the team and management, and between management and the client; the discipline to adhere to strict standards and conventions; and responsibility. She believes all of these points to be crucial, and that the importance of each increases with the overall size of the task.

It would appear that those same ingredients contributed to the success of the Forth programmers and scientists who worked on experiments performed aboard space shuttle missions in November 1981 and October 1984. Dr. Henry Harris of Pasadena's Jet Propulsion Laboratory described his work with JPL and the Johnson Space Center. The three-year project focused on instrument control and, in the end, was beset by in-orbit hardware problems that would have been insurmountable if it were not for Forth's interactiveness.

JPL chose to use Forth for this project because of its adaptability to a wide range of needs and conditions. They employed intense number-crunching, graphics and sophisticated arrays of geometrical attitudes, comprising something like a megabyte of compiled code. The team implemented elements of LISP and Prolog in their LMI Forth system, which was then used to develop a context-sensitive editor and critical constraint checking to define permissible instrument movements.

Dr. Harris asserted that, because of its modularity, Forth excels in large programs. It is good for jobs with critical deadlines (such as a launch date) and where interactive control can help meet changing conditions (as when it was used to save the shuttle experiment during hardware failures). Harris then presented three myths that have been dispelled by recent accomplishments: that Forth is only good for small programs; that no good software has ever been written in Forth; and that if a program is larger than 64K, it isn't right. By way of one example, JPL's Forth on an IBM-XT more accurately predicted the space shuttle's position than did the mainframe at Johnson Space Center.

All told, the Rochester Forth Conference provided a platform for about sixty speakers to address an audience of nearly two hundred. Several described their experiences with creating significant extensions to Forth for purposes of education, artificial intelligence and adding object orientation, for example. It became apparent that Forth is a natural seedbed for the best features of contemporary computer languages, which can be easily implemented in Forth without the possible restrictions of the environment in which they originate. Charles Duff discussed NEON, an object-oriented language making waves in the greater Macintosh community; Pierre Moreton presented HFORTH, an "English-like business application language"; and Arnold Epstein intrigued listeners with his MAGIC/L, which incorporates a Pascal-like syntax for ease of maintainence.

For an afternoon session, the conference divided into about a dozen working groups, special-interest gatherings ranging in size from six to twenty members. Topics included education, Forth under Unix, robotics, image processing and artificial intelligence, state machines, Forth in hardware, standards and others. Brief reports from the groups indicated a variety of progress, with at least one working group determining to continue research and to report back at next year's conference.

One evening was devoted to a tour of the University of Rochester's Laboratory for Laser Energetics, with whose cooperation the conference was held. For two hours, conference attendees inspected the Forth-controlled fusion research facility, whose multiple-beam laser can be fired at half-hour intervals. LLE workers were kind enough not only to tolerate the presence of more than one hundred visitors crowding an evening shift, but to answer many questions and generally humor us.

The lecture program concluded with

a presentation by Lawrence Forsley, conference chairman. His paper on "nth-Order Defining Words" discussed clearly a subject that is somewhat abstract for many, the definition of words that will define other words. Although limited by time, Larry also managed — with the help of an assistant — to demonstrate programming on the fly, and the difficulties of debugging an overhead projector. Needless to say, the audience was delighted.

The final day consisted of a number of well-attended exhibits, including Forth Inc., Miller Microsystems, Forth machines from Novix and Metaforth, the Institute for Applied Forth Research, the Forth Interest Group and Dash-Find Assoc., a Forth-specific employment referral service. Concurrent tutorials attracted members of the community as well as conferees.

In addition to the caliber of presentations, integral to the success of the Rochester conference was the inspiration and efficiency of its organizers. Congratulations and thanks go to Lawrence P. Forsley, conference chairman; Thea Martin, conference coordinator; and to Jonathan Ross and Sherry Snyder of the institute. Their friendly and patient efforts were a reward to all who attended.

—Marlin Ouverson

For details of the approximately sixty presentations, watch for the published Proceedings of the 1985 Rochester Forth Conference.

Ultra-Fast Text Search Using

Probabilistic Dictionaries

John James
Santa Cruz, California

Suppose you have a large database of text files, such as the documents in a lawyer's office, perhaps one or 100 megabytes or more in dozens or hundreds of separate files. You want to search the whole database for any files which contain certain words, phrases, or AND and OR combinations of words and phrases. And you want the search to be very fast, almost instantaneous.

I haven't implemented this procedure, and don't know of anyone who has. So it's impossible to know for sure how well it will work. Despite the uncertainty, this tutorial seems worth publishing because it illustrates important data structures and concepts as well as offering the promise of important practical benefits.

Background:
The Probabilistic Dictionary

Used in some spelling checkers, the probabilistic dictionary is a most interesting data structure, and one which opens new approaches to a number of problems. I'll explain it first with an example. Suppose a spelling checker needs a dictionary of about 40,000 English words but only has 100K bytes to store them in RAM and can't afford the speed penalty of keeping the words on disk, which could require a separate disk access for each word being checked. A probabilistic dictionary can comfortably store the words in two-and-a-half bytes each, no matter how long the actual words are.

To build the dictionary, start by clearing all the 100K bytes (800K bits) to zero. Now, to add each word to the dictionary, use ten different "hashing" functions (see below for explanation of hashing) on the ASCII strings which represent the word. Each hashing function will select one of the bits, by computing a number between zero and 799,999 from the ASCII string. Set each of the ten selected bits to "1" — whether or not it had been set before.

After all 40,000 English words have been entered in this way, somewhat fewer than 400,000 bits — half of the 800,000 available — will have been set. Now, to look up a word, compute the ten hash functions on it and see if all ten bits are set. If even one of the bits is not set, then the word is definitely not in the dictionary. If all ten of the bits are set, the word is probably in the dictionary. But with each bit having about a fifty percent probability of being set anyway, there is about one chance in a thousand of a false hit, meaning that a word not in the dictionary would be "found." For a spelling program, this represents a one-in-a-thousand chance that a misspelled word would not be detected, a risk which may be acceptable. Hence the name "probabilistic dictionary."

Background: Hashing

The point of "hashing" is to take a value, such as an English word or a lengthy part number, and transform it into an arbitrary number which is a more suitable key for indexing into a table. In the example, we need to transform an English word into a number from 0 to 799,999. Of course, different words may transform into the same number (a "collision"); but we try to select a computational procedure such that the numbers don't tend to bunch up, so there won't be too many collisions.

A common hashing computation is to divide by an appropriate number, throw away the quotient and take the remainder. In the example above, this divisor would be a little less than 800,000, so that the remainder will be the right size to select one of the bits. The divisor should be a prime number; for more background on picking a good one, see Knuth[2]. In the above example, ten different divisors would be used to select the ten bits in the table of 800,000. Note that before division, the English word whose ASCII string will be the dividend should be left-justified or rearranged in some other way; otherwise, for short words the dividend would be less than 800,000 and all ten divisions would give the same remainder. Picking a good hashing function can be tricky.

Dictionary for Text "Search"

Now let's use the probabilistic dictionary for another purpose — text searching. The trick is to prepare a separate, small dictionary for each file in the database. All these little dictionaries might fit into RAM or in a small file on disk. Then, to search for a word in the entire text database, just look it up in all of the dictionaries. Only one pass through the dictionaries is enough, even for a complex search involving many words, phrases, and AND and OR combinations of them.

Different kinds of condensed dictionaries could be used, not only the probabilistic dictionary suggested here. We suggested this one because it is compact, easy to implement and very efficient, especially when many dictionaries must be searched for the same words.

How big should each of the dictionaries be? A typical text document of about 10,000 words will have only about 1500 distinct words; a 3000 word dictionary would accommodate much more than a 20,000 word document. Two-and-a-half bytes per word gives 7.5K bytes for each document's dictionary. And note that each dictionary can actually hold more than 3000 words and in fact will never overflow, although performance degrades gradually if too many words are added. The actual dictionary size can be selected by the end user, although for any given database of text files, it's easier to keep all dictionaries the same size than to have larger ones for large documents.

If 500K of RAM is available, over sixty dictionaries of 7.5K each could be kept there for faster searches, although it wouldn't hurt much to keep these dictionaries as a file on disk, since a search would need only one pass through that file. Each dictionary should also contain the name of the file which it represents. Then the search through the dictionaries could yield a list of file names. These files would then be searched word by word (full text search), both to eliminate the few false hits resulting from the probabilistic nature of the dictionary, and also to locate the words or phrases sought, in case the user wanted to examine them in context.

Miscellaneous Hints

(1) Logical AND and OR searches are easily handled, but NOT presents a problem because it changes false hits, which are tolerable, into false misses, which are not. So use the full text search before excluding any file from consideration.

(2) Searches are fast. The hash functions need only be computed once, no matter how many dictionaries need to be searched, so the time to compute them should be imperceptible. And for any single word or AND of words, a single bit mask can be prepared for high-speed test against the dictionaries. It's reasonable to estimate that the dictionary search of dozens of files, even long ones, could be completed within a second or two. Any full text searches required would take longer, but usually they can be concurrent with display of the output to the user.

(3) The suggested 7.5K-bytes dictionary size requires hash-function divisors around 60K. If your system doesn't have an unsigned division available, consider getting one bit by another method, such as adding the ASCII letters of the English word together and taking the last bit of the sum. But note that this bit will be the same for all ten hash values of a given word.

(4) Phrases can be handled as an AND of the words, with false hits excluded later in the full text search.

(5) This method won't find parts of words. But common prefixes and suffixes can be treated. For example, to avoid missing plurals of words, just drop the final "s" or "es" before entering or searching for any word. Some false hits will occur, but they will be resolved during the full text search, at a modest cost in performance. Let users know that irregular plurals must still be searched explicitly, with an AND.

(6) In the full text search, you can do better than looking at every letter. For example, when you are searching text for a word, look at the last-letter position in the text first. A quick lookup in a small table specially prepared for the word being sought will usually show that that particular letter is not contained anywhere in the word; and in that case, the intermediate character positions don't need to be examined at all, since they could not possibly be part of the word being sought. Many other refinements have been developed, since text searching has such practical importance.

(7) Include each file's last-modification date and time with its dictionary. Then the search program can examine the directory of files, to automatically regenerate the dictionary for any new or changed text file. This mechanism will also generate the dictionary initially, the first time the program is run.

References

1. Bentley, Jon, "A Spelling Checker," *Communications of the ACM*, May 1985, Vol. 28, No. 5. This "Programming Pearls" article includes background on text documents and on several methods of checking spelling, and has good references for further research.

2. Knuth, Donald E., *The Art of Computer Programming*, Vol. 3, Addison-Wesley, Reading, Mass., 1973. This classic programming text has a long section on hashing and related methods.

• ALABAMA

Huntsville FIG Chapter
Call Tom Konantz
205/881-6483

• ALASKA

Kodiak Area Chapter
Call Horace Simmons
907/486-5049

• ARIZONA

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

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

• ARKANSAS

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

• CALIFORNIA

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

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

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

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

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

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

Bay Area Chapter
Silicon Valley Chapter
Monthly, 4th Sat.,
FORML 10 a.m., FIG 1 p.m.
ABC Christian School Aud.
Dartmouth & San Carlos Ave.
San Carlos
Call John Hall 415/532-1115
or call the FIG Hotline:
408/277-0668

Stockton Chapter
Call Doug Dillon
209/931-2448

• COLORADO

Denver Chapter
Monthly, 1st Mon., 7 p.m.
Call Steven Sarns
303/477-5955

• CONNECTICUT

Central Connecticut Chapter
Call Charles Krajewski
203/344-9996

• FLORIDA

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

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

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

• GEORGIA

Atlanta Chapter
Call Ron Skelton
404/393-8764

• ILLINOIS

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

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

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

Rockwell Chicago Chapter
Call Gerard Kusiolek
312/885-8092

• INDIANA

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

Fort Wayne Chapter
Monthly, 2nd Wed., 7 p.m.
Indiana/Purdue Univ. Campus
Rm. B71, Neff Hall
Call Blair MacDermid
219/749-2042

• IOWA

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

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

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

• KANSAS

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

• LOUISIANA

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

• MASSACHUSETTS

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

• MICHIGAN

Detroit Chapter
Monthly, 4th Wed.
Call Tom Chrapkiewicz
313/562-8506

• MINNESOTA

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

• MISSOURI

Kansas City Chapter
Monthly, 4th Tues., 7 p.m.
Midwest Research Inst.
Mag Conference Center
Call Linus Orth
816/444-6655

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

• NEVADA

Southern Nevada Chapter
Call Gerald Hasty
702/452-3368

• NEW HAMPSHIRE

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

• NEW MEXICO

Albuquerque Chapter
Monthly, 1st Thurs., 7:30 p.m.
Physics & Astronomy Bldg.
Univ. of New Mexico
Call Rick Granfield
505/296-8651

• NEW YORK

FIG, New York
Monthly, 2nd Wed., 8 p.m.
Queens College
Call Ron Martinez
212/517-9429

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

Rockland County Chapter
Call Elizabeth Gormley
Pearl River
914/735-8967

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

• OHIO

Athens Chapter
Call Isreal Urieli
614/594-3731

Cleveland Chapter
Call Gary Bergstrom
216/247-2492

Cincinatti Chapter
Call Douglas Bennett
513/831-0142

Dayton Chapter
Twice monthly, 2nd Tues., &
4th Wed., 6:30 p.m.
CFC 11 W. Monument Ave.
Suite 612
Dayton, OH
Call Gary M. Granger
513/849-1483

• OKLAHOMA

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

• OREGON

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

• PENNSYLVANIA

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

• TENNESSEE

East Tennessee Chapter
Monthly, 2nd Tue., 7:30 p.m.
Sci. Appl. Int'l. Corp., 8th Fl.
800 Oak Ridge Turnpike, Oak Ridge
Call Richard Secrist
615/693-7380

• TEXAS

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

Dallas/Ft. Worth
Metroplex Chapter
Monthly, 4th Thurs., 7 p.m.
Call Chuck Durrett
214/245-1064

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

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

• UTAH

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

• VERMONT

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

• VIRGINIA

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

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

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

• WISCONSIN

Lake Superior FIG Chapter
Call Allen Anway
715/394-8360

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

FOREIGN

• AUSTRALIA

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

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

• BELGIUM

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

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

• CANADA

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

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

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

• COLOMBIA

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

• ENGLAND

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

• FRANCE

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

• GERMANY

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

• IRELAND

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

• ITALY

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

• JAPAN

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

• REPUBLIC OF CHINA

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

• SWITZERLAND

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

SPECIAL GROUPS

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

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

FIGGRAPH
Call Howard Pearlmutter
408/425-8700

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

BULK RATE
U.S. POSTAGE
PAID
Permit No. 3107
San Jose, CA

Address Correction Requested