FORTH DIMENSIONS

VOLUME IV, NUMBER 5 $2.50

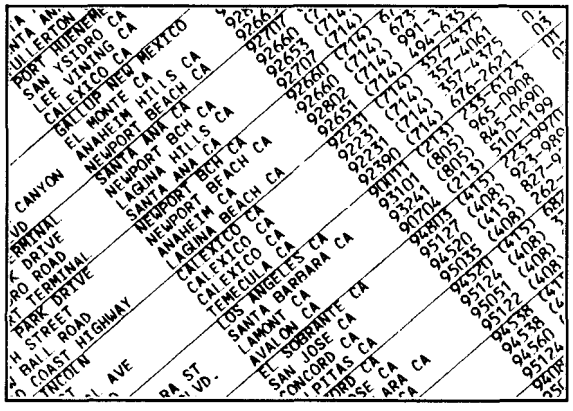
INSIDE:

BUSINESS APPLICATIONS

FORTH in Business Applications ..... Panel Discussion 4
The Indexer: Enhancements to a Data Base Model ..... Robert N. Watkins, CCP 10
A PICTURE Worth a 1000 Words ..... Elmer W. Fittery 19

DEPARTMENTS:

Letters 3
Standards Corner 29
A Techniques Tutorial 26



BUSINESS APPLICATIONS

THE FORTH SOURCE™

MVP-FORTH – A Public Domain Product

MVP Forth is fig-FORTH modified by 100% of the FORTH-79 Standard Required Word Set plus the vocabulary for the instructional book *Starting FORTH*. Editor, assembler and utilities are included.

Transportability of programs is assured since the kernal of MVP-FORTH is the same for all computers to the machine dependent READ/WRITE instructions.

Modification and extension (up or down) is simplified by having the source code and through the use of MVP-FORTH Programming Aids and Cross Compilers.

The CP/M® are supplied on 8", SS/SD, IBM 3740, format disks. The include a track and sector calculation array for down loading to other sizes and formats. Other disks are machine specific.

All About FORTH is an annotated glossary of MVP-FORTH words as well as other dialects. It is in 8080 code, other MVP-FORTH implementations include documentation of the differences between it and other CPU's and computers.

MVP-FORTH PRODUCTS for CP/M® IBM-PC® and Apple®

- ☐ **MVP-FORTH Programmer's Kit** including disk with documentation, ALL ABOUT FORTH, and STARTING FORTH. Assembly source listing versions. $100
- ☐ **MVP-FORTH Disk** with documentation. Assembly source listing version. $75
- ☐ **MVP-FORTH Cross Compiler** with MVP-FORTH source in FORTH. $300
- ☐ **MVP-FORTH Programming Aids** for decompiling, callfinding, and translating. Specify computer. $150
- ☐ **MVP-FORTH Fast Floating Point** for Apple II/II+ on board with 9511 math chip. Requires MVP-FORTH for Apple $400
- ☐ **MVP-FORTH Assembly Source** Printed listing. $20
- ☐ **ALL ABOUT FORTH** by Haydon. MVP-FORTH reference, plus fig-FORTH and FORTH-79. $20

●★★MVP-FORTH operates under a variety of CPU's, computers, and operating systems. Specify your computer and operating system. CP/M supplied on 8", SS/SD, 3740 format.●★●

FORTH DISKS

FORTH with editor, assembler, and manual.

- ☐ **APPLE II/II+** by MicroMotion $100
- ☐ **APPLE II** by Kuntze $90
- ☐ **ATARI®** valFORTH $50
- ☐ **CP/M®** by MicroMotion $100
- ☐ **CROMEMCO®** by Inner Access $100
- ☐ **HP-85** by Lange $90
- ☐ **IBM-PC®** by Laboratory Microsystems $100
- ☐ **NOVA** by CCI, quad floppy $100
- ☐ **PET®** by FSS $90
- ☐ **TRS-80/I®** by Nautilus Systems $90
- ☐ **6800** by Talbot Microsystems $100
- ☐ **6809** by Talbot Microsystems $100
- ☐ **Z80** by Laboratory Microsystems $50
- ☐ **8086/88** by Laboratory Microsystems $100
- ☐ **VIC FORTH** byHES, VIC20 cartridge $60

Enhanced FORTH with: F-Floating Point, G-Graphics, T-Tutorial, S-Stand Alone, M-Math Chip Support, MT-Multi-Tasking, X-Other Extras, 79-FORTH-79.

- ☐ **APPLE II/II+** by Micro-Motion, F, G, & 79 $140
- ☐ **ATARI** by PNS, F, G, & X $90
- ☐ **CP/M** by MicroMotion, F & 79 $140
- ☐ **Apple II/II+, GraFORTH** by Insoft, stand alone graphics $75
- ☐ **H89/Z89** by Haydon, T & S $250
- ☐ **H89/Z89** by Haydon, T $175
- ☐ **IBM-PC, PolyFORTH** by FORTH Inc., F, G, S, M, MT, & X $300
- ☐ **Multi-Tasking FORTH** by Shaw Labs, CP/M, X & 79 $395
- ☐ **TRS-80/I or III** by Miller Microcomputer Services, F, X, & 79 $130
- ☐ **TUTORIAL** by Laxen & Harris, CP/M with a copy of *Starting FORTH* $95
- ☐ **Extensions** for Laboratory Microsystems IBM,Z80, and 8086
  - ☐ Software Floating Point $100
  - ☐ 8087 Support (IBM-PC or 8086) $100
  - ☐ 9511 Support (Z80 or 8086) $100
  - ☐ Color Graphics (IBM-PC) $100
  - ☐ Data Base Management $200

CROSS COMPILERS Allow extending, modifying and compiling for speed and memory savings, can also produce ROMable code. •Requires FORTH disk.

☐ CP/M	$300	☐ IBM•	$300
☐ H89/Z89	$300	☐ 8086•	$300
☐ TRS-80/I	$300	☐ Z80•	$300
☐ Northstar®	$300	☐ Apple II/II+	$300

☐ **fig-FORTH Programming Aids** for decompiling, callfinding, and translating. Specify CP/M, IBM-PC, 8086, Z80, or Apple II/II+ $150

FORTH MANUALS, GUIDES & DOCUMENTS

- ☐ **ALL ABOUT FORTH** by Haydon. An annotated glossary of common FORTH words. MVP-FORTH reference. $20
- ☐ **And So FORTH** by Huang. A college level text. $25
- ☐ **FORTH Encyclopedia** by Derick & Baker. A complete programmer's manual to fig-FORTH with FORTH-79 references. Flow charted, 2nd Ed. $25
- ☐ **Starting FORTH** by Brodie. Best instructional manual available. (soft cover) $16
- ☐ **Starting FORTH** (hard cover) $20
- ☐ **1980 FORML Proc.** $25
- ☐ **1981 FORML Proc.** 2 Vol. $40
- ☐ **1982 FORML Proc.** $25
- ☐ **1981 Rochester FORTH Proc.** $25
- ☐ **1982 Rochester FORTH Proc.** $25
- ☐ **Using FORTH** $25
- ☐ **A FORTH Primer** $25
- ☐ **Threaded Interpretive Languages** $20
- ☐ **AIM FORTH User's Manual** $12
- ☐ **APPLE User's Manual** MicroMotion $20
- ☐ **TRS-80 User's Manual,** MMSFORTH $19
- ☐ **METAFORTH** by Cassady. Meta compiler in 8080 code $30
- ☐ **Systems Guide to fig-FORTH** $25
- ☐ **Caltech FORTH Manual** $12
- ☐ **Invitation to FORTH** $20
- ☐ **PDP-11 FORTH User's Manual** $20
- ☐ **CP/M User's Manual,** MicroMotion $20
- ☐ **FORTH-79 Standard** $15
- ☐ **FORTH-79 Standard Conversion** $10
- ☐ **Tiny Pascal in fig-FORTH** $10
- ☐ **NOVA fig-FORTH** by CCI with editor, assembler, and utilities $15
- ☐ **MVP-FORTH** Source Listings ☐ IBM-PC ☐ CP/M ☐ Apple II/II+ $20

☐ **Installation Manual for fig-FORTH,** contains FORTH model, glossary, memory map and instructions $15

Source Listings of fig-FORTH, for specific CPU's and computers. The Installation Manual is required for implementation. Each $15

☐ 1802	☐ 6502	☐ 6800	☐ AlphaMicro
☐ 8080	☐ 8086/88	☐ 9900	☐ APPLE II
☐ PACE	☐ 6809	☐ NOVA	☐ PDP-11/LSI-11
☐ 68000	☐ Eclipse	☐ VAX	

Ordering Information: Check, Money Order (payable to MOUNTAIN VIEW PRESS, INC.), VISA, MasterCard or COD's accepted. No billing or unpaid PO's. California residents add sales tax. Shipping costs in US included in price. Foreign orders, pay in US funds on US bank, include for handling and shipping by Air: $5 for each item under $25, $10 for each item between $25 and $99 and $20 for each item over $100. Minimum order $10. All prices and products subject to change or withdrawal without notice. Single system and/or single user license agreement required on some products.

DEALER & AUTHOR INQUIRIES INVITED

MOUNTAIN VIEW PRESS, INC.

PO BOX 4656 **MOUNTAIN VIEW, CA 94040** **(415) 961-4103**

Letters . . .

Helpful Documentation

Dear FIG,

Your correspondent, Derek Vair, echoed my problems. I had recently tried to tread the same path. Mountain View press sorted me out very efficiently and courteously, even though by miscalculation I phoned them at 3:00 a.m. one night. I cannot speak too highly of Glen Hayden's "All about FORTH." It is a work of meticulous scholarship and surely a must for all FORTH novices such as myself.

Dennis A. Larder
Saudi Arabia

Another 1 4 2

Dear FIG,

I write to you on behalf of the 'TO-solution.'

For more than two years I've programmed instrumentation systems and microprocessors in FORTH. I have experienced a reluctance of my customers to accept FORTH, which stems to a great deal from the seemingly awkward notation of **A @ B @ + C !** compared to C = A + B, which everybody is used to. And I really think that FORTH code would become clearer, if an explicit assignment operator like **=** would exist.

We do have this operator, called **TO** by Bartholdi. Ever since reading his paper I have included the construct into my system, but I prefer to call the operator **=:**. Thus the above code reads **A B + =: C**.

A, **B** and **C** are variables of the new datatype **INTEGER**. There have been discussions in F.D., whether **INTEGER**s ought to replace **VARIABLE**s. I prefer to use both datatypes, each in the place where it fits best (i.e. makes code clearer).

Judging by published programs in F.D. and the conference reports, many people seem to use **TO**. My experience with the **TO** or **=:** solution is so good that I wonder, why it is not considered for inclusion into the 1983 standard.

Thank you for your efforts and for the good work being done by FIG.

Wolf Wejgaard
Switzerland

Importing Annnouncement

Dear FIG,

We have noticed among your correspondents various commercial vendors of FORTH hardware/software. As recent and enthusiastic converts, we would be interested in establishing contact with vendors to see if we can import and distribute various FORTH-oriented products in the U.K. Perhaps you could publish this letter so that interested vendors may contact us directly.

A. Williams
Ledbury Electronics Ltd.
Unit 2, Little Marcle Lane
Ledbury, Herefordshire HR8 2AU

Don't Leave FIGs Hanging

Dear FIG,

The new editorial policy of *Forth Dimensions* of giving preference to FORTH-79 programs, and tutorials using FORTH-79 is very reasonable, but a little troublesome for those who have implemented FIG's 8080 model. I have recently implemented this model on a Swedish microcomputer and the implementation will be distributed to our club's 2,000 members within the coming month. I would like to make it easier for them to learn and use FORTH via Brodie's book and *Forth Dimensions*, so I should suggest an alternative vocabulary which defines FORTH-79 behavior in the necessary instances. Could you suggest a source for the information I need to construct this alternative vocabulary?

A revision of the 8080 model has been in the works for at least a year. I assume that it will follow the FORTH-79 standard. If this version is soon to be released, it would be better for me to implement the new version and thus eliminate the conflict of definitions between fig-FORTH and FORTH-79. Can you tell me if the new 8080 (Z80 ?) version will soon be released?

Thank you for your help, on behalf of the ABC80 club.

Dr. Robert Johnsen
Uppsala, Sweden

Our official guide to conversion from the FIG model to 79-Standard is a document by Robert L. Smith called FORTH-79 Standard Conversion. *It is available for $10.00 (U.S.) from Mountain View Press.*

We're not aware of any current movement on the part of the FORTH Interest Group towards a Standard revision of the model. We feel that this is the purvue of the FORTH vendors. One such vendor, which not only offers a 79-Standard FORTH but has also placed it in the public domain, is Mountain View Press. —Editor

FORTH Dimensions

Published by FORTH Interest Group
Volume IV, No. 5
January/February 1983

Editorial/Production
Leo Brodie

Publisher
Roy C. Martens

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/or to submit material, the address is: FORTH Interest Group, P.O. Box 1105, San Carlos, CA 94070

FORTH in Business Applications
A Panel Discussion

Moderated by John Hall
4th Annual
FORTH National Convention
October 9, 1982
Transcribed from tape by
Ray Van de Walker

John Hall, Moderator: Welcome to our panel discussion on Business Applications. Our first panel member is Mitch Derick, a consultant with Concepta. He has a large FORTH business application that he would like to talk about. Our next panel member will be Dick Miller of Miller Microcomputer Services. He has some actual products and has been working for quite a while with business applications. Marvin Benedick, president of Soft-Tax, will be discussing his tax-preparation package. I chose him because he has learned FORTH, produced a product, and has apparently found FORTH to be perfect for his application. Fourth on the panel is Charles Moore, the inventor of FORTH, who will be asking the question, "Is FORTH really good for business applications?"

Mr. Derick?

Derick: I'd like to talk about a large business application but first I think that I should tell you something about our situation. I've been working with FORTH for 3½ or 4 years. We got into FORTH slowly at first but now we use only FORTH. We've worked for a company called Abacus II for the last two years. We've done everything from systems analysis down to the programming, testing, and even some maintenance, because this is a long project.

The project is a fault-tolerant computer system, both hardware and software, to be installed in fast-food restaurants. The first restaurant that we are aiming for is McDonalds, so we are trying to satisfy a big, sophisticated market. As a result, we've produced a big, sophisticated product.

In this system everything is written in FORTH — the operating system, the fault-tolerant code, the utilities, the data base and the applications. The system has two basic functions. The first is the point-of-sale portion which is what you see the clerk using to enter your order. The point-of-sale portion is used to handle the on-line functions such as cash, coupons, gift-certificates, what products are sold, and any other things that normally occur up-front during a sales transaction.

The data from the point-of-sale function is used by the other major portion of the system, the manager function. The manager function is a very friendly interface between the managers, who are unsophisticated users, and the computer's applications. In real-life, the user may be a 14 or 15 year-old kid who was told to do something by the manager.

So what we have is a pressure-sensitive transparent touch-screen over the face of the CRT which displays standard McDonald's forms, with which the users are supposed to be familiar.

The total system handles inventory, usage of ingredients, wasted materials (a cook may drop a hamburger), shipments in, shipments out, transfers, ordering projections (How much did the store use last week? How much last year at this time of the year?), a very comprehensive statistical package for the mother corporation (McDonalds tracks all materials to four decimal places, even including the ketchup usage) and personnel (Who works in the store. Is he under 16 years old? Is he a high-school graduate? What skills does he have?). All of that information is necessary for crew scheduling which can be a major problem since the average store has about 300 employees, mostly part-time.

As you can imagine there's a definite need in the marketplace for this product. The problem with similar products in the past was simply that the computers didn't work. By coming into the marketplace with a fault-tolerant system, we should do quite well.

The total package consists of about 50 applications, all written in FORTH. This size of project is significant compared with some others that I've seen. I sat down and figured it out and over the last 2½ years we've written roughly 2500 screens of code. From the standpoint of a traditional programming environment, say a fantastic government data base project in a large COBOL shop, this is a medium-sized project.

A project of this sort is quite common in the data processing industry and could have been done in most any language. But we didn't do it in just any language, we did it in FORTH. And because we did it in FORTH, we got some benefits that are, realistically, only available in FORTH. Any of these things could have been done in other languages but they never actually would be because it would be too inconvenient. Only FORTH and its extensibility would have given us the benefits that we've gotten.

Now I'd like to talk about these benefits and how they helped the project to be successful. Something to keep in mind is that Abacus II is a start-up company, doing research and development in new fault-tolerant techniques. The company's financial position was basically very insecure and complicated by the necessity of having to show results to financial people throughout the false starts and dead-ends that always accompany R&D.

This environment is where FORTH started showing its true colors. In this sort of environment what does a company really need? It doesn't have a product yet so it has no income, so what it needs most is money and it has to get that money by impressing financial people with a demo that shows the potential of the company. Abacus II spent a lot of time meeting milestones, showing continual progress to these people by means of demos. This approach led to a concept of vertical design and implementation. I can best explain this by looking at a traditional language first. At the beginning of a three year project you do system analysis and way over at the end, after three years, you release a salable product. In between you do specification, design, coding, testing and validation. It is a sequential process and it takes the entire life of the project. There's no time for error. If you make a wrong turn near the beginning of the project, you may not find out until customer verification, three years later.

What you do in vertical design and implementation is that you slice your project vertically. Then you implement a very high level design and bring in the financial people and say, "Look, this is what we're trying to do." Then these people can sit down and interact with a live system. They can touch the screen, call up dummy forms and generally see what we're trying to do.

We were able to gradually replace the simulation stubs by increasingly detailed operational code. At each milestone we were able to give a demo consisting of actual code that would be part of the final system. We weren't just writing demos, we were doing useful programming as well. After about three weeks we had something that we could show. Now, this could be done in other languages, but it's just not convenient.

In this sort of environment you can sit down with a customer and find out: is this feature good?; this looks fine; that doesn't, and so forth. On the next pass he might say: "Wow, you fixed that, but it's not what I meant. What I really think would be better would be this." You just keep iterating until it's right.

We even changed computers in midstream. Just the other day we finished moving the code from a 6502 to a Z80 system. This is the benefit of standardization. After we factored out the terminal drivers and some other peripheral-dependent code, the rest of the system just worked. I specifically checked on this. The parts that were just FORTH ran — no problems, no trouble — they just ran.

Now, this is a large complex product and that means that it has a long product life. Since it has a long product life it will need to be maintained for years. Some people estimate that maintenance costs can be up to 90% of a product's total cost! So in order to produce a maintainable system we wrote the system with good style.

Some critics have characterized FORTH as a "write-only" language but that claim is absolutely wrong. These applications are the closest to English of anything that I've ever seen. We didn't use FORTH like a 250 mnemonic assembler. What we did was create English. We could take an owner-operator of a McDonalds store, sit him down to look at the FORTH code, and he could read it. The high and intermediate-level code is written in their terms rather than in computerese or some programmer's terms.

Basically, we did this with factoring — one function to a module. Probably the best rule of thumb has turned out to be that if you have a comment in a module, you throw away the old name and the parentheses and make the comment the new name of the module. This quick rule of thumb made readable code.

The proof of the pudding has been maintenance — it's been long enough to see some results. Originally we had two schools of thought — short, cryptic words save memory and long English names will make maintenance easier. The cryptic code had to be thrown out. The English code is being maintained by junior-level programmers who, in my experience, could not have done the job in other languages. These are some of the benefits of using FORTH.

There's one more key point that has proven to be very important. When we first started we had some experienced programmers but mostly we had junior level programmers. We had very few programmers who had ever used FORTH. We had Kim Harris come in and give a class in FORTH. By giving the class we eliminated the learning curve that normally goes with making full use of the features of FORTH. Because FORTH is different, there's a learning curve. Learning curves are expensive. This course eliminated that expense.

FORTH was just a marvelous language to work in immediately. I can heartily recommend that if you plan a large project, get your programmers educated.

So in summary I'd have to say that for commercial applications, FORTH is the only language for me.

Moderator: Do we have any questions?

Audience: Are there about 50 modules in your application?

Derick: No, there are about 50 applications — the number of modules is larger than that and still growing.

Audience: You say that you've targeted McDonalds as the market for your system. Do owner-operators have the freedom to buy directly from you or do you have to sell these systems to the whole chain?

Derick: There's a thing called the "Approved Vendor List." At the moment no one is on it. The people who were on it before were removed. They were using 1970 state-of-the-art equipment, and it's 1982 now. We have to deal with both the giant corporation and the individual owner-operator.

Audience: Will McDonalds like FORTH because it's 20% smaller? (laughter)

Derick: No, headless FORTH is closer to being 30% smaller. Seriously, we chose McDonalds because it's a huge potential market.

Moderator: I'm sorry to interrupt, but we really must move on to our next panelist, Dick Miller of Miller Microcomputer Services.

Miller: Is FORTH practical for business? "Clearly, it isn't!" is what we hear from a lot of people. Most recently in fact from the special FORTH issue of *InfoWorld* that was out this week. I won't blame *InfoWorld* because there were a lot of quotes in there that said that FORTH is great for process control and bad for business. There are a lot of reasons for that. One is it doesn't have strings or any capabilities in that direction and another is that it is totally unreadable, as we have already discussed.

FORTH seems to have been working for us for business for 4 or 5 years. We built our own FORTH system. We might not have but there was no other way to get a FORTH system for the computers we were working on at the time. Of course once we had a FORTH system, we sat down and started to play. Then we played for money and discovered that FORTH wasn't good enough for that. FORTH is a tool not a solution. FORTH programs which get hung up on a stack error are not too re-assuring for secretaries and will probably give FORTH a bad name. FORTH is fine for programmers. That's never been the problem. What's needed are programmers understanding what users are. We have some pretty good ideas on that.

My wife Jill is a full partner in the company and has 15 years of experience with large industrial business applications. She knows that FORTH is good and that little computers are often better than big ones. You can also put the little computers where you want them. Once you have them, you have to make them talk intelligently.

The big problem there is that people who spend $2,000 for the computer don't know how much they ought to spend for programs. Usually these people are certain that it should be a lot less! For some reason the applications don't seem to be too *available* at $15 each. We hit that problem in several ways. We didn't even talk to people who were using small computers, at first. Instead we talked to people who were using big computers and told them how to do some of their jobs on little computers. They had the budget and understanding and were happy to

Continued

save money on the hardware while also saving some money on the software. They weren't begrudging us the fact that the application might be costing them as much as the computer.

This situation is going to be very clear by next year — that we can't bring the cost of people down as quickly as we can the cost of computers. So we got ourselves a tool, an amplifier — FORTH.

We've done a lot of the expensive kind of programming that we've just described and it's been very successful. The string handling is aboard in our FORTH, as it is today in most real FORTHs. To us and the user, the strings look about like those in BASIC and they run a lot more directly. We see the disk when we want to, and the user sees a files directory when that's appropriate and doesn't when it's not appropriate.

We have a full list of applications that are available inexpensively to go with our system. We didn't do this because we're benevolent but because we needed them for our other professional projects. Bit by bit, we developed a library of applications and then went back and documented them for casual users. We have a full communications package. The package will seem full to most of you but a few of you may want more. But, as I heard in a previous panel, designing something that's everything for everybody may take an infinite amount of time and the package probably will not fit in 64K. We have a pretty complete system targeted for most real applications. We customize like mad for the other kind of applications.

We have a data base system that's rather well known. It trades off large relational data base capabilities for real-time results. It's very powerful and we think that it's the best professional tool for over half of the real-world professional applications. We know it isn't for the other portion, the applications which must answer big fancy questions in one shot.

When we need more capability we customize it in, as do others who buy the package from us. We have a word processor called ForthWrite and I like it a lot. We've used it for many manuals and we're pleased with some of its new tricks, one of which is to take data from DataHandler, the data base package, and turn out customized form letters. If you want, you can list all of our Congressmen and send them customized letters like they send to you.

We have a general ledger package that out performs the minicomputer packages that we analyzed for the company that consulted us for advice on which package to buy. We next got the rights to move the Osborne accounting package into FORTH. We weren't smart enough about accounting, so the accountants who worked with us threw us and the package out on our ear and we had to come back with a package that worked. We had to stop thinking like programmers and find out what the job really was and then come back to it as programmers. Our system is very fast and it makes the people who use it about twice as efficient as with comparable packages because they spend most of their day typing on the numeric keypad and they don't have to look at anything except the work that they are typing in. Considerations of efficiency are important for serious users.

We have all these packages available as standard packages which run on our standard system, which is available on several computers. We also have special requests. Sometimes that means special hardware, sometimes a modification for the pre-existing hardware, and sometimes it means starting from scratch.

We find our packages easy to modify because certain building-block words can be factored out and changed. For example, our communications package which normally runs menus at 1200 baud can be easily modified to run 9600 baud on the TRS-80, or over three times that on the IBM-PC. We've done some very interesting jobs just moving bits and pieces of these packages around and splicing them together. The strange thing is that we can afford to sell the packages complete for less money than we can sell their pieces. We're selling popular applications. The applications cost a lot to make but if we can sell enough of the same application, we do OK and the customer does OK.

The big shortfall of FORTH that I see these days is that FORTH is not being delivered to users with many usable applications. That's what we've been trying to do. We think that FORTH is good enough and that there's a big enough market out there to pay for good things. Our little applications are optimized to show results on the screen instantly. We can't do everything that way but the things that we can do that way are so useful that they have found a market.

Sometimes we have to trade off that speed for fancier processing or else get fancier hardware, but we're willing to do those things. If you're looking for an application that will do well, find an application that people need and then trade off its capabilities so that 90% of the people are happy about how you traded off.

If you have an application that you need to develop in two months, look around for tools that already exist to help you finish the job. If you have trouble getting started, hire some professionals to come in and write the most difficult parts of the job while simultaneously training your group. By using this method your group will understand the problem in time to finish the job. FORTH works if you handle it reliably and these methods are some fairly reliable cookbook recipes that work.

Our applications are written in 79-Standard FORTH and if you don't have one of our machines, we are happy to sell you the application words that you can pop over into someone else's 79-Standard FORTH. So, understand what the 79-Standard means when you have a big job to get done on time. There's a reason why FORTH has been confusing and part of that reason has been that FORTH has been different from one installation to the next. So think standard. The missing link in the past has been that users had to go a long way off to get the applications that they needed. What we've been trying to do is to sell an application that is *almost* right for someone in your neighborhood, for a few hundred dollars. In that way, you can modify the package for a few more hundred dollars and produce what the customer wants. This is the sort of work that a lawyer or doctor does. A few of them are inventing new diseases, cures or statutes, but most of them are helping someone to get a problem fixed. So, our non-custom work is intended to be 90% of your custom work and we believe that FORTH is a good vehicle for this sort of work.

Moderator: Could we have some questions now?

Audience: How can you support a programming environment in which the customers have customized systems?

Miller: We only support what we sell. For example, we don't run under other operating systems, although we do have a FORTH that runs under CP/M. We call it the shoehorn and we don't sell it, we only use it for in-house development. It's a lousy FORTH that lets us get into the machine and bring over a good FORTH.

Audience: What I mean is, when you sell a customized system, how do you support it?

Miller: We differentiate between clients and customers. A customer gets a debugged, standardized package. A client pays us by the hour for a different level of service. For our clients we maintain a listing with the modifications from the standard package marked in green. If the modification is small it fits in our standard blocks. If the modification is large, then the standard block loads the modification from somewhere else and the load is marked in green. Large modifications also have a beginning block which has a little code and a lot of text. We can usually find out very quickly what differences our client's systems have. I suppose though that the real trick is not to customize unless you need to. The way to do that is factoring. Factor out the parts which remain unchanged from the standard utility.

Moderator: Thank you Dick. Our next speaker is Marvin Benedick of Soft-Tax.

Benedick: I'd like to give some background about my company. As the name implies, I write an application that's used by commercial tax preparers. My programming experience has been 5 years with North Star BASIC and 15 months with FORTH.

My application has about 25 subprograms, 500 "fill-in-the-box" style inputs and 100 supporting screen displays. The application is fairly large but very repetitive because it fills in tax forms. FORTH excels in this repetitive string and number-handling environment. In BASIC the program was 350K. In FORTH its size was reduced to 140K. This 210K saving was largely caused by the repetitive re-use of subroutines. The routines for adding and subtracting dates are an excellent example of routines whose re-use saved much repetitive coding. In BASIC the date routines had to be re-written for each application. By re-using one set of date routines I was able to save much time in debugging. Any savings of time is very important because, besides writing the program, time is my worst enemy. I receive my final tax law proofs in late November and I must have my program written, debugged and documented by January 1. When I was trying to meet this time schedule in BASIC I found that it required several programmers.

In FORTH we are able to update not only the program but also our documentation which is also written in FORTH. In our documentation we felt that a picture is worth a thousand words, so many of our screen displays are also re-used to produce illustrations. In FORTH we were able to re-use both our screen displays and input sheet codes to produce camera-ready copy for our printer. This procedure saved me many hours of cut and paste work and saved me from needing to use someone else's word processing program. Essentially, my documentation is now a FORTH program. With FORTH I am now able to read my programs a year later. I have been able to reduce my program-modification period because my routines have meaningful names now instead of numbers. I believe that what I am doing does not even scratch the surface of FORTH's capabilities. Starting on July 1, 1981, I was able to learn FORTH and rewrite a 350K BASIC program, incorporating new tax-law changes and an improved design, in time for my January 15, 1982 release date. I am very impressed with FORTH as a language — I think that it's the language for me. Any questions?

Audience: Is this a package that you sell, or a service?

Benedick: It's a package. It runs under both Northstar and CP/M. I sell it to enrolled tax preparation agents and CPA's to speed up their work. It helped that FORTH was so portable.

Audience: Is your documentation a help-type facility or are you using FORTH to print out your documentation?

Benedick: My documentation is a series of FORTH words which control the printer to produce documentation when they are executed. My documentation really is a FORTH program. I don't have a help facility because my programs are already too big. I couldn't afford to have that much text.

Moderator: Thanks Marvin. Our next speaker is Charles Moore. I guess he doesn't need any other introduction.

Moore: The question is often asked "Is FORTH really good for business?". It's as if FORTH's apparent virtues will somehow fall short in this area and some subtle problem will rule it out. Well, I'd like to answer the question with a qualified yes. FORTH is really good for business. But I don't know that business people know or want to know that.

I have been speculating over the last year about motives. It's not clear that your typical manager of a corporate computer center wants to minimize the size of the program, computer and staff, while maximizing the program's maintainability and response speed. These are things that FORTH is good at. He loses at every stage — he wants the biggest, most expensive installation he can have because that increases his salary, stature and influence within the company. Rather than saying that's wrong, it's necessary to recognize that it's so. To try to sell such a person on the virtues of FORTH is foolish.

To achieve some market penetration it may be useful to divide the business world into large business and small business. Small businesses have traditionally relied upon their accountants and lawyers to provide the services that a computer now provides. FORTH would make it more possible for a computer to provide those services. Nevertheless, it may be more fun or more effective to deal with a trained person than with a computer. Therefore, perhaps even in the small business area, FORTH is not good for business.

If a business has management which is willing to be incompatible with everything (because FORTH really is incompatible with everything else) for the sake of economy and efficiency and if that management is willing to innovate and accept the hassle that new computer systems always have, then FORTH is an excellent tool for that sort of management.

I'm pleased to see that Dick, Mitch and Marvin mentioned readability. FORTH is truly a readable language. I've heard for ten years that "FORTH is very nice, but it's not a readable language." I have achieved a state of mind in which I don't even wince, although I might say that the person doesn't understand the problem. There is no way that I can convince a person that FORTH is readable short of teaching him to read it. The question is not the degree of FORTH's readability but rather whether people are willing to learn to read it.

It is folklore that an effective manager knows more than the people he is managing. A manager of programmers is wise to use his programmers to write code but he is also wise to read the code that his programmers write. If he doesn't, he places himself in a very vulnerable position because he is accountable, not the programmers. The ability of the manager to read the code implies that the programmers must write readable code. Anyone can write UNreadable code. The presence of funny little at-signs and exclamation points (now pronounced "fetch" and "store" respectively) can be minimized with a certain style of programming. In business applications

Continued

where readability is paramount you shouldn't use them. They should not be removed from the language because they are useful — but their use should be minimized when the code will be read by non-programmers.

It is almost always possible to arrange the FORTH words that describe an application in such a way that the words form meaningful English sentences. FORTH gives you a great deal more freedom in doing things than almost any other language. For example, in my business package I have a word called "941", which generates the form for a 941 tax report. Fortran would never allow such a subroutine name. Somehow, I feel that no one using that application will ever type 941 without meaning that particular report.

I hate to write off the large business world because there is a lot of fun to be had by solving the many corporate problems that occur. One of my day dreams has been to reprogram IBM's corporate books. I think that they could probably be placed on a minicomputer with improved throughput and greater executive confidence in the results. I think that the project would be about one man-year. Of course, I've just been hanging back, waiting to be invited . . . (laughter) but I no longer think that it will happen.

There have been a number of large applications that have been done in FORTH. Any "start-up" application that will run on a dedicated stand-alone machine is a good candidate for FORTH because there's no installed conservative attitude. Big inventory systems have been done in FORTH, banking systems have been done in FORTH, accounting systems, stock market "back rooms," even hospital systems.

There's no reason to fear a large application — the tools for handling a large application are more available in FORTH than in any other language. The challenge is to implement these applications with fewer programmers than they would otherwise require. Programmers will have to handle larger parts of the program but this should lead to happier programmers. Programmers like to solve significant parts of a problem. Letting a person handle the entire accounts receivable section for example, can be a powerful incentive to get that person working for you.

I have always wanted to put FORTH on a mainframe. I have always thought that a 370 (or whatever they're called now) with hundreds of terminals would be a challenging environment. With the passage of time mainframes have become not only obsolete but uninteresting. We are now in the position where it is probably easier to build the mainframes out of minicomputers than it is to program the mainframes. A network of microcomputers is cheaper, easier to program, maintainable, fail-soft, redundant, etc. All the good words apply so strongly that I think that the mainframes will quietly rust away in the back rooms while a network of micros quietly take over the real work in the front rooms. That's about all I have to say.

Moderator: Any questions? Any general questions?

Audience: I read in *Business Week* that many companies are forming what they call "decision centers" because the conventional data processing facility has too slow a turn-around to be effective in making decisions. Do any of you know of this?

Dick Miller: I have a comment. We do a lot of business with large companies but very little of it is through their data processing centers. What's been happening is that every corner of the corporation except the data processing department is quietly getting a "desk-top data processor," or some other device that doesn't sound quite like a "computer." It allows them to get their answers in an half hour instead of two weeks later. The smart data processing organizations are beginning to talk to us. Often they seem to figure out a system where the little computer talks to the big computer, leaving the big computer more free for the large tasks and eliminating many of the complaints.

Audience: Why do you think FORTH is particularly well suited to distributed processing?

Charles Moore: There are two reasons. First, much of distributed processing consists of a large number of simple tasks. FORTH is usually very good at providing large quantities of simple things. Whether the system consists of many tasks running on a single computer, or even more tasks each with its own copy of the machine and software, is only a small difference. The other reason is that FORTH is very, very good with communication protocols of all kinds. A simple serial communication usually only requires two lines of code at each end. Layered protocols, node addresses and other such complexities are usually handled rather nicely by FORTH and very poorly by the only other language which can handle network communication, namely assembly language.

Moderator: Well, our time has run out. Thank you gentlemen.

Moderator John Hall is a FORTH consultant in the Oakland, California area. □

THE INDEXER:
Enhancements to a Data Base Model

Robert N. Watkins, CCP
Long Beach, California

A previous article in *FORTH Dimensions* by Glen Haydon (3) presented a good foundation for data base work. This article describes the addition of an indexing scheme and query language to that design to implement a specific type of data base optimized for retrieval of data records by keywords.

The essence of the previous article is contained on screens 61 and 62. The eight FORTH words defined there enable the declaration of a data base file, fields within each logical record in such a file, and some primitive operations.

Each file is described by a file information block, or FIB, which is created by the defining word **FILE**. A FIB (see figure 1) holds the starting block number of the file, the maximum number of records it may contain, the length of a physical record (buffer) and the length of a logical record, several of which may fit into a physical record.

Invoking the name of a file places the address of its FIB into a variable called **OPEN**. All other access words use **OPEN** to point to the FIB of the currently open file. The word **READ** places a logical record number (relative to zero) at the address returned by **REC#**, to indicate which logical record in the file is to be considered current.

The defining words **DFIELD** and **TFIELD** can be used to create words that describe fields within the logical record. Invoking these words will cause the disk block containing the current record to be loaded if needed and return the starting address of the field within the disk buffer. Words created with **TFIELD** also leave a count on the stack for use by words like **TYPE** and **CMOVE**.

Given the basic file development words above it is possible to construct

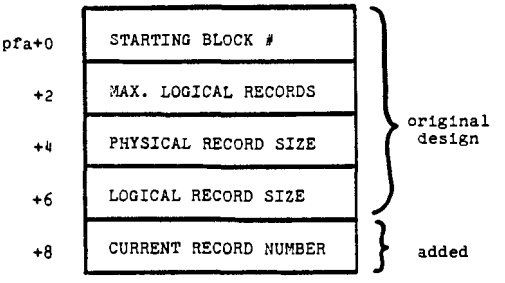


Figure 1. File Information Block

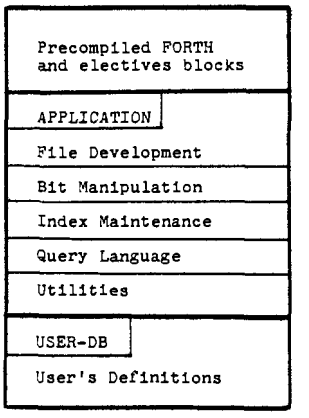


Figure 2. Dictionary Layout

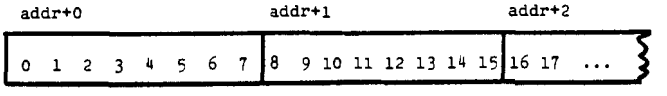


Figure 3. Bit number designations

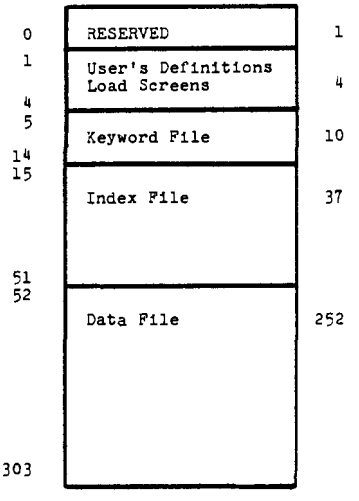


Figure 4. Example Diskette Allocation

many different visions of a data base. The Indexer is merely one such vision. My purpose was to find a way to cross reference data records under several subject headings and be able to conduct queries to extract subsets of the file. For example, if I ran a clipping service and found an interesting article on bicycling, I would want to be able to quickly find out which of my customers would be interested in a copy.

The goals were: (1) fast response time on reasonably complex queries; (2) reasonable response time on file maintenance; and (3) tolerable use of disk space. The method I chose is based on a manual method of indexing using optical incidence cards (1). Numbers are assigned to the keywords and the appropriate numbers are punched into each card. By superimposing cards and holding them up to the light, matches can be spotted. The Indexer uses bitwise logical operations to superimpose index records and bit testing to "hold them up to the light."

Each Indexer data base consists of four disk areas which are load screens, keyword file, index file and data file. Think of a little kid (L'KID) to remember them.

The load screens contain the user's definitions of fields in the data base (see screens 27 and 28 of the previous article). Also, any words to manipulate those fields, such as **.REC** to list the contents of the current record, could be contained here.

The keyword file contains the subject terms that are to be used in indexing. In the version contained in this article, each can be up to 32 characters long and have embedded spaces. The keyword file is updated much the same as any other file.

The index file is essentially a bitmap: for each combination of keyword and data record there exists one bit which is set to one to indicate a relationship or left zero to show no relationship. For each keyword there is a corresponding index record. For each data record there is a bit in each index record. This file is maintained by words that establish a linkage between whichever data file record is current and the desired keywords.

Continued

```
SCR # 1
  0 ( INDEXER Case Study DB - Field Definitions )
  1 FORGET USER-DB      : USER-DB ;
  2 0 30 TFIELD NAME   ( Name of contact )
  3   16 TFIELD PHONE ( Phone number of contact )
  4   16 TFIELD NETWORK ( Network address of contact )
  5
  6 : !NAME  ( -- )  ( input and store contact name )
  7    NAME 32 FILL    QUERY 1 TEXT    PAD COUNT
  8    NAME ROT MIN    CMOVE   UPDATE    ;
  9 : .NAME  ( -- )  ( print name field )  NAME TYPE ;
 10
 11 : !PHONE  ( -- )  ( input and store contact phone # )
 12    PHONE 32 FILL    QUERY 1 TEXT    PAD COUNT
 13    PHONE ROT MIN    CMOVE   UPDATE    ;
 14 : .PHONE  ( -- )  ( print phone field )  PHONE TYPE ;
 15    -->


SCR # 2
  0 ( INDEXER Case Study, cont'd      Field def'ns cont'd     )
  1
  2 : !NETWORK  ( -- )  ( input and store network address )
  3    NETWORK 32 FILL    QUERY 1 TEXT    PAD COUNT
  4    NETWORK ROT MIN    CMOVE   UPDATE    ;
  5 : .NETWORK  ( -- )  ( print net address )  NETWORK TYPE ;
  6
  7    -->


SCR # 3
  0 ( INDEXER Case Study, cont'd      Utility words       )
  1 : .REC  ( -- )  ( print contents of current record )
  2    CR    .NAME    .PHONE    .NETWORK    CR CR    ;
  3
  4 : NEW-DATA  ( -- )  ( add new data record to DB )
  5    DFILE    @RCNT    1+ DUP    !RCNT    READ
  6    CR    ." Enter contact name: "    !NAME
  7    CR    ." Enter telephone #: "    !PHONE
  8    CR    ." Enter network address: "    !NETWORK
  9    .REC    FLUSH    ;


SCR # 60
  0 CR ." THE INDEXER:  Loading ... "
  1
  2 FORGET APPLICATION          : APPLICATION ;
  3
  4 ( First load the database package published by FIG )
  5 61 LOAD    62 LOAD     ( originally screens 22 and 23 )
  6 63 LOAD    64 LOAD    65 LOAD    ( originally 24-26 )
  7
  8 ( Now load The Indexer's screens          )
  9 : LOADEM    83 66 DO    FORTH I LOAD    LOOP    ;  LOADEM
 10
 11 ( All done.  Tell user how to proceed )
 12 CR ." Ready.  Insert your database disk and type: "
 13 CR ."       LOAD-DB "       ;S


SCR # 61
  0 CR ." SCREEN 61: FILE DEVELOPMENT  (22) "
  1 0 VARIABLE OPEN    ( points to current file block )
  2 : REC#    OPEN @    8 +    ;  ( holds current record # )
  3
  4 ( 2@  DUP 2+ @ SWAP @ ;  ( double fetch )
  5
  6 : LAYOUT  ( leave bytes/record-2, bytes/block-1 )
  7    OPEN @ 4 + 2@ ;
  8 : READ    ( n-th record, on stack, is made current )
  9    0 MAX DUP OPEN @ 2+ @ < 0=
 10    IF ." FILE ERROR " QUIT THEN REC# ! ;
 11 : RECORD    ( leave address of n-th record )
 12    LAYOUT */MOD OPEN @ @ + BLOCK + ;
 13 : ADDRESS  ( leave address of current record )
 14    REC# @ RECORD ;
 15  ;S
```

Listing continued on next page

```
SCR # 62
  0 CR ." SCREEN 62: FILE DEVELOPMENT - 2  (23) "
  1 : TFIELD <BUILDS  ( create a text field )
  2    OVER , DUP , + ( leaves file count for this defn )
  3          DOES>  ( leaves addr, count )
  4    2@ ADDRESS + SWAP ;
  5 : DFIELD  <BUILDS  ( create a data field )
  6    OVER , +        ( leaves file count for this defn )
  7           DOES>  ( leave address )
  8    @ ADDRESS + ;
  9 : FILE             ( create a named storage allocation )
 10    <BUILDS  ,     ( starting block in file )
 11      1+ ,         ( number of records in file )
 12      DUP B/BUF OVER / * , ( #bytes / block )
 13    , 0 ,       ( bytes/record, current rec# )
 14     DOES> OPEN ! ;  ( when file name is used, point to )
 15    ;S              ( its descriptor block )

SCR # 63
  0 CR ." SCREEN 63: DOUBLE PRECISION ARITHMETIC  (24) "
  1
  2 : D/   ( d  u -- d   )
  3    SWAP OVER /MOD >R SWAP U/ SWAP DROP R> ;
  4 : D*   ( d  u -- d   )
  5    DUP ROT * ROT ROT U* ROT + ;
  6 : D/MOD  ( d  u -- d   )  U/  ;
  7 : D@     ( addr -- d   )  DUP 2+ @ SWAP @ ;
  8 : D!     ( d addr -- )  DUP ROT SWAP ! 2 + ! ;
  9 : DDUP   ( d  -- d d  )  2DUP ;
 10 : DSWAP   ( d1 d2 -- d2 d1 )  ROT >R ROT R> ;
 11 : DDROP   ( d1  d2 -- )   DROP DROP ;
 12    ;S

SCR # 64
  0 CR ." SCREEN 64: DATE COMPRESSION AND EXPANSION  (25) "
  1
  2 : DATEBIT ( converts date input to 2 bytes )
  3    32 D* D+ 13 D* D+ DROP  ;
  4
  5 : ?DATE     ."  ( MM/DD/YY ) "
  6    QUERY 47 WORD HERE NUMBER 47 WORD HERE NUMBER BL WORD
  7    HERE NUMBER DATEBIT ;
  8
  9 : .DATE     ( FORMAT 2 BYTE DATE CODE AND PRINTS )
 10    0 13 U/ 32 /MOD ROT 100 * ROT + 0 100 D* ROT 0 D+
 11    <# # # 47 HOLD # # 47 HOLD # # #> TYPE ;
 12   ;S

SCR # 65
  0 CR ." SCREEN 65: ?$AMOUNT AND .$AMOUNT  (26) "
  1 ( define action for each scale case )
  2 : 0SCALE 100 D* ;   : 1SCALE 10 D* ;   : 2SCALE ;
  3 : 3SCALE ." INPUT ERROR " CR ;
  4 ( define scale case and extend for each with 'CFA' )
  5 ' 0SCALE CFA VARIABLE NSCALE ' 1SCALE CFA , ' 2SCALE CFA ,
  6 ' 3SCALE CFA ,
  7 ( scale double precision value according to 'DPL' )
  8 : SCALE DPL @ 3 MIN 2 * NSCALE + @ EXECUTE ;
  9 ( wait for decimal value and scale it - leave value on stack )
 10 : ?$AMOUNT QUERY BL WORD HERE NUMBER SCALE ;
 11 ( print d from stack as $ and right justify in 8 spaces )
 12 : .$AMOUNT
 13    DUP ROT ROT DABS <# # # 46 HOLD #S SIGN #>
 14    36 EMIT DUP 8 SWAP - SPACES TYPE ;
 15  ;S

SCR # 66
  0 CR ." SCREEN 66: BIT MANIPULATION "
  1 : >AND  ( from to count -- )  ( move while ANDing )
  2    2*  0  DO
  3                2DUP I + @ SWAP
  4                I + @ AND
  5                OVER I + !
  6          2 +LOOP  2DROP ;
  7
  8 : >OR  ( from to count -- )  ( move while ORing )
  9    2* 0 DO
 10             2DUP I + @ SWAP
 11             I + @ OR
 12             OVER I + !
 13    2 +LOOP  2DROP ;
 14
 15    ;S
```

The data file is the same as in the previous article, with one addition: the last n positions of the record are reserved for a list of keyword numbers that are associated with the record. (n = 2 x the maximum number of keywords/data record.) This redundancy has speed benefits at maintenance time, as the bitmapped index has speed advantages at inquiry time. In addition, the keywords list enables the index file to be rebuilt for the data file if a system crash or other mishap causes the two files to be out of synchronization with each other.

Figure 2 shows the dictionary layout of the Indexer. After FORTH is loaded, my system's last electives screen defines the null word **APPLICATION**. The Indexer **FORGET**s back to that word, then re-defines it. The original article's screens come next, followed by Bit Manipulation, Index Maintenance, Query Language and Utility words. Finally another null word, **USER-DB**, is defined to provide a **FORGET**ing boundary for the user's data base definition.

The earlier data base screens were used with only one modification: the variable **REC#** was replaced by a word which uses **OPEN** to place the current record number within the FIB. The Indexer switches among the three files in the file set and the current record number of each must be preserved.

To accomplish this I reversed the order in which **REC#** and **OPEN** were defined on screen 61. Then I redefined **REC#** in terms of **OPEN**, to return addr+8 of the currently open file's FIB. Finally I added a "zero comma" to the **<BUILDS** portion of the defining word **FILE** to make room for the record number in each FIB and initialize it to zero. The FIB layout modification is shown in figure 1.

This modification to **REC#** illustrates the functional nature of FORTH (2). Recall that a variable leaves on the stack an address at which the data can be found. The new definition has the same stack effect (interface to other words) yet internally is implemented quite differently to extend the concept of the current record.

Screens 63-65 are the date and dollar conversion routines from the previous article. No changes were made.

Screens 66-68 contain bit manipulation routines. **>AND**, **>OR**, and **>NOT** are designed to be similar to the '79 Standard **MOVE** in that they move 16-bit cells at a time and take three arguments from the stack: from-addr, to-addr and cell-count. But a funny thing happens on the way to the to-address — the desired logical opration is performed bitwise on the cell.

An excellent article in *Byte* magazine about a disk operating system (4) inspired the bitmap words that follow. **^ 2** ("carat two" or "uparrow two") raises 2 to the top-of-stack power. **BITMASK** takes a given bit# and produces the mask required to test that bit; if the bit number is greater than seven, the address on the stack is also adjusted to point to the correct byte address containing the bit. See figure 3 for bit# designations. **BITON** and **BITOFF** do exactly that — turn a given bit on or off. **?BIT** returns the status of a given bit.

Specifying both an address and a bit# in these three words gives flexibility. You can number bits from the start of the logical record, or restrict the bit numbers to the range 0-7 and work with individual bytes.

Screen 69 contains working variables. **WORKSIZE** is a constant that tells how large the current working set is in bytes. **WORKSET** is that working set — a bitmap with one bit per data record in the file. If a bit is on, the associated data record is a member of the subset under consideration. **WORK#** holds the current bit# within the working set.

The next four words are used to keep track of the current query — that set of commands that created the working set. **#STEPS** tells how many commands have been given. **QVERB** and **QOBJ** are tables (maximum capacity 30 entries) that store the number of the command given and the number of the keyword involved, if any. **"VERBS"** is a superstring that explains what the various command numbers mean, for use by **.QUERY**.

The last three words on screen 69 are the file designations. As in the previous article, defining a file requires three arguments: the logical record length, maximum logical records, and starting block number for

Continued

```
SCR # 67
  0 CR ." SCREEN 67: BIT MANIPULATION - 2 "
  1 : >NOT  ( from to count -- )  ( move, complementing )
  2    2* 0 DO
  3           OVER   I +   @   COMPLEMENT
  4           OVER   I + !
  5       2 +LOOP   2DROP   ;
  6
  7 : A2  ( n -- 2^n )  ( see ref [1] )
  8    DUP 0=  IF DROP 1
  9            ELSE 1 SWAP 0 DO 2* LOOP
 10           THEN ;
 11
 12 : BITMASK  ( bit# -- bitmask )  ( see text )
 13    8 /MOD   ROT + SWAP   7 SWAP - A2  ;
 14    ;S

SCR # 68
  0 CR ." SCREEN 68: BIT MANIPULATION - 3 "
  1
  2 : BITON  ( addr bit# -- )  ( set indicated bit = 1 )
  3    BITMASK   OVER C@   OR   SWAP C!  ;
  4
  5 : BITOFF  ( addr bit# -- )  ( set indicated bit = 0 )
  6    BITMASK COMPLEMENT   OVER C@   AND   SWAP C!  ;
  7
  8 : ?BIT  ( addr bit# -- f )
  9    BITMASK   SWAP C@   AND   0= NOT  ;
 10    ;S

SCR # 69
  0 CR ." SCREEN 69: WORKING VARIABLES "
  1
  2 128 CONSTANT WORKSIZE  ( working set bitmap in bytes )
  3 0 VARIABLE WORKSET   WORKSIZE 2- ALLOT
  4 0 VARIABLE WORK#  ( current record # in working set )
  5 16 VARIABLE KWD/REC  ( max. keywords per data record )
  6 0 VARIABLE #STEPS   ( length of current query )
  7 30 1ARRAY QVERB   30 1ARRAY QOBJ
  8 : "VERBS"  ( superstring with verb literals )
  9    ." NONE;  ALL;   SELECT(AND(   OR(    ANDNOT(ORNOT(  " ;
 10
 11  32  300  5 FILE KFILE  ( keywords file )
 12 128  300 15 FILE IFILE  ( index bitmap file )
 13 256 1000 52 FILE DFILE  ( data file )

SCR # 70
  0 CR ." SCREEN 70: PRE-DEFINED FIELDS "
  1 0 2 DFIELD RCNT DROP      ( all files: current rec. count )
  2 0 32 TFIELD KWD DROP      ( keyword file: only field   )
  3 0 2 DFIELD RECSTART DROP  ( all files: rec start addr )
  4 : @RCNT  ( -- n )  ( get record count for open file )
  5    OPEN @   8 + @   0 READ   RCNT @   SWAP   READ   ;
  6 : !RCNT  ( n -- )  ( store updated count for open file )
  7    OPEN @   8 + @   SWAP   0 READ   RCNT !   UPDATE   READ ;
  8 : !KWD  ( -- )  ( input & store keyword in record )
  9    KWD 32 FILL   QUERY 1 TEXT   PAD COUNT
 10    KWD ROT MIN   CMOVE   UPDATE   ;
 11 : NEW-KWD  ( -- )  ( add record to keyword file )
 12    KFILE   @RCNT   1+ DUP   !RCNT   READ
 13    CR   ." ENTER NEW KEYWORD: "   !KWD
 14    CR   KWD TYPE   FLUSH   ;

SCR # 71
  0 CR ." SCREEN 71: INDEX MAINTENANCE "
  1 : ?NULL  ( -- f )  ( true if PAD has a null string )
  2    PAD DUP   1+ C@ 0=   SWAP   C@ 0=   OR   ;
  3 : +NDX  ( data# kwd# -- )  ( update index for kwd )
  4    OPEN @ ROT ROT   IFILE READ   RECSTART SWAP BITON
  5    UPDATE   OPEN !   ;
  6 : -NDX  ( data# kwd# -- )  ( update index for kwd )
  7    OPEN @ ROT ROT   IFILE READ   RECSTART SWAP BITOFF
  8    UPDATE   OPEN !   ;
  9 : ?KWD  ( -- kwd# )  ( kwd# if defined, 0 if not )
 10    0   OPEN @   KFILE @RCNT
 11    1+ 1 DO   ( loop thru keywords until found or not )
 12       FORTH I READ
 13       PAD COUNT KWD DROP -TEXT   ( compare )
 14       IF   SWAP I + SWAP   LEAVE   THEN
 15    LOOP   OPEN !   ;
```

Listing continued on next page

```
SCR # 72
  0 CR ." SCREEN 72: INDEX MAINTENANCE - 2 "
  1
  2 : LISTLIM  ( -- limit start ) ( calc limits for kwd list )
  3    RECSTART   OPEN @ 6 + @    +   2-  ( start addr )
  4    DUP   KWD/REC @   2* -   SWAP   ;
  5
  6 : ?SLOT  ( n -- addr )  ( returns addr of matching slot or 0 )
  7    0 SWAP          ( return value if loop expires w/o match )
  8    LISTLIM     ( set limits of keywords list in data rec )
  9    DO
 10       I @   OVER =       ( set kwd# from list & compare )
 11       IF   SWAP DROP I SWAP LEAVE THEN  ( return the addr )
 12    -2 +LOOP   DROP   ;

SCR # 73
  0 CR ." SCREEN 73: INDEX MAINTENANCE - 3 "
  1 : +KWD  ( -- )  ( update all linkages for 1 keyword )
  2    OPEN @   DFILE REC# @   DUP   0=
  3    IF ."  -- NO CURRENT RECORD " DROP OPEN ! QUIT THEN
  4    ?KWD   DUP   0=
  5    IF ."  -- NO SUCH KEYWORD " 2DROP OPEN ! QUIT THEN
  6    DUP ?SLOT          ( is the kwd already in the list )
  7    IF 2DROP           ( yes, no further updating req'd )
  8    ELSE  0 ?SLOT  DUP   ( no, is there room for one more? )
  9       NOT IF ." -- TOO MANY KEYWORDS " DROP 2DROP OPEN ! QUIT
 10       ELSE   >R DUP R> ! UPDATE   +NDX
 11       THEN
 12    THEN
 13    OPEN !     ;  ( restore the previously open file )

SCR # 74
  0 CR ." SCREEN 74: INDEX MAINTENANCE - 4 "
  1 : -KWD  ( -- )  ( remove all linkages for 1 keyword )
  2    OPEN @   DFILE REC# @   DUP   0=
  3    IF ."  -- NO CURRENT RECORD " DROP OPEN ! QUIT THEN
  4    ?KWD   DUP   0=
  5    IF ."  -- NO SUCH KEYWORD " 2DROP OPEN ! QUIT THEN
  6    DUP ?SLOT DUP  ( is the kwd already in the list )
  7    IF   >R 0 R> !   UPDATE   -NDX   THEN
  8    OPEN !     ;  ( restore the previously open file )
  9
 10 : +TERMS  ( -- )  ( add n keywords to current data record )
 11    BEGIN   44 TEXT   ?NULL NOT   ( delimiter is comma )
 12    WHILE   CR PAD COUNT TYPE   +KWD   REPEAT   ;

SCR # 75
  0 CR ." SCREEN 75: INDEX MAINTENANCE - 5 "
  1
  2 : -TERMS  ( -- )  ( remove n keywords from current record )
  3    BEGIN   44 TEXT   ?NULL NOT
  4    WHILE   CR PAD COUNT TYPE   -KWD   REPEAT   ;
  5
  6 : .TERMS  ( -- )  ( print keywords for current data record )
  7    LISTLIM DO
  8             RECSTART DROP   ( keep data rec in buf. via BLOCK )
  9             I @             ( get keyword #, possibly zero )
 10          IF   OPEN @   KFILE I @ READ
 11               CR KWD -TRAILING TYPE   OPEN !
 12          THEN
 13       -2 +LOOP   ;

SCR # 76
  0 CR ." SCREEN 76: QUERY LANGUAGE "
  1
  2 : .QVERB  ( index -- )  ( print verb[index] )
  3    QVERB @
  4    7 *   ' "VERBS"   3 +   +   7  -TRAILING TYPE  ;
  5
  6 : .QOBJ  ( kwd# -- )  ( print keyword name )
  7    QOBJ @   DUP 0=   IF   2 SPACES   ELSE
  8    OPEN @ SWAP  KFILE READ  KWD -TRAILING TYPE
  9      ." )"   OPEN !  THEN   ;
 10
 11 : .QUERY  ( -- )  ( print query that created working set )
 12    #STEPS @ 1+   1   DO
 13    CR   I .QVERB   SPACE   I .QOBJ   LOOP ;
```

the file. The selections here waste very little space on my Micropolis Mod II drive: on each 5.25" diskette I have 304 blocks divided as shown in figure 4.

It is not necessary to contain the entire data base on a single diskette as I have done. If you have more than one drive you can split the files among different drives to make more room or to make the keywords file common to several data bases. In a bartering service for example, the same keyword file could serve both Wanted and Offered data bases. Robert F. Cramer suggests that indexing is most effective when this is the case (1). By defining the starting blocks correctly the various files could appear anywhere within your total disk environment. A user with a hard disk would need to alter the contents of the various FIBs in the load screens to prevent conflicts among multiple data bases on the disk.

Screen 70 contains pre-defined fields for the files in the data base. **RCNT** is used with record number zero in keyword and data files and holds the current record count of the file. **KWD** is the single field in keyword records. **RECSTART** is used by all files to give the starting address of the logical record. The convention of **!** for store and **@** for fetch is used in **@RCNT**, **!RCNT** and **!KWD**.

A new convention was required to replace the word **INPUT** in the previous article since we need to update both the keyword and data files directly. I use the word **NEW-KWD** to serve as **INPUT** for the keyword file, and **NEW-DATA** (in each data base's load screens) to do the same for the data file.

Screens 71-75 define words used to update the index. They require particular attention because most of the file switching and other tricky code is found here. I adopted a convention that either a word would take responsibility for saving on the stack which file was open when it started, then restore it before leaving, or it would ignore such things altogether and operate transparently on the currently open file. The sequences **OPEN @** and **OPEN !** as a matched set indicate that the word will be switching the open file.

After a user has entered the text of a keyword in conjunction with an index maintenance or query word it is placed at **PAD**. **?NULL** checks to see if the string entered is null. **+NDX** takes a tuple of data record number and keyword number and establishes the relationship in the index by turning on the proper bit. **−NDX** does the opposite. **?KWD** uses the contents of **PAD** to search the keywords file and returns either the keyword number or zero if no match was found.

To update the list of keywords in the data record, **LISTLIM** provides a range of addresses that can be used as **DO** loop limits. **?SLOT** uses that range plus a keyword number and returns the address at which the keyword number is found in the list or zero if not found. **+KWD** and **−KWD** bring it all together, doing error checking and updating both the data and index records for one keyword. But the only words the user needs are **+TERMS** and **−TERMS**, which take multiple keywords separated by commas and either add them to the current data record or remove them. **.TERMS** simply lists which keywords are associated with the current data record.

Screens 76-81 contain the query language words. A query is a list of commands that affect the working set. **.QVERB** and **.QOBJ** are used by **.QUERY** to print out all the steps in the current query, one per line.

NONE; and **ALL;** fill the working set with zeroes or ones respectively. **?KARG** is used by the remaining query words to accept a keyword from the user and determine if it is valid, returning keyword number if so and quitting if not.

The next five words have left parenthesis as part of their names to remind the user that a keyword is required after them. **SELECT(** sets the current working set equal to the index record of a given keyword. **AND(** performs a logical **AND** with the working set and the index of the named keyword. **OR(** does the same thing with the logical **OR** operation. **ANDNOT(** and **ORNOT(** are similar to **AND(** and **OR(** except that they complement the index record bitmap before doing the **AND** or **OR**. Since

Continued

```
SCR # 77
  0 CR ." SCREEN 77: QUERY LANGUAGE - 2 "
  1
  2 : NONE;  ( -- )  ( make working set empty )
  3    WORKSET WORKSIZE 0 FILL    ( set all bits off )
  4    1 #STEPS !                 ( set query to beginning )
  5    0   #STEPS @   QVERB !  ( put verb 0 at line 1 )
  6    0   #STEPS @   QOBJ !   ;  ( no kwd for this verb )
  7
  8 : ALL;  ( -- )  ( select all data records )
  9    WORKSET WORKSIZE 255 FILL   1 #STEPS !
 10    1   #STEPS @   QVERB !   0   #STEPS @   QOBJ !   ;
 11
 12 : ?KARG  ( -- kwd# )  ( returns valid keyword# or quits )
 13    41 TEXT   ?NULL   IF ."  -- NO KEYWORD GIVEN " QUIT THEN
 14    ?KWD   DUP 0=   IF ."  -- NO SUCH KEYWORD "   QUIT THEN ;

SCR # 78
  0 CR ." SCREEN 78: QUERY LANGUAGE - 3 "
  1 : SELECT(  ( -- )  ( set working set to one keyword )
  2    ?KARG   OPEN @ SWAP   DUP IFILE READ
  3    RECSTART WORKSET WORKSIZE CMOVE
  4    1 #STEPS !   2 #STEPS @ QVERB !   #STEPS @ QOBJ !
  5    OPEN !   ;
  6 : AND(  ( -- )  ( logical AND working set with keyword )
  7    ?KARG   OPEN @ SWAP   DUP IFILE READ
  8    RECSTART WORKSET WORKSIZE 2/ >AND
  9    #STEPS @ 1+ #STEPS !   3 #STEPS @ QVERB !
 10    #STEPS @ QOBJ !   OPEN !   ;
 11 : OR(  ( -- )  ( logical OR working set with keyword )
 12    ?KARG   OPEN @ SWAP   DUP IFILE READ
 13    RECSTART WORKSET WORKSIZE 2/ >OR
 14    #STEPS @ 1+ #STEPS !   4 #STEPS @ QVERB !
 15    #STEPS @ QOBJ !   OPEN !   ;

SCR # 79
  0 CR ." SCREEN 79: QUERY LANGUAGE - 4 "
  1 : ANDNOT(  ( -- )  ( AND working set with NOT of keyword )
  2    ?KARG   OPEN @ SWAP   DUP IFILE READ
  3    RECSTART PAD WORKSIZE 2/ >NOT
  4    PAD WORKSET WORKSIZE 2/ >AND
  5    #STEPS @ 1+ #STEPS !   5 #STEPS @ QVERB !
  6    #STEPS @ QOBJ !   OPEN !   ;
  7
  8 : ORNOT(  ( -- )  ( OR working set with NOT of keyword )
  9    ?KARG   OPEN @ SWAP   DUP IFILE READ
 10    RECSTART PAD WORKSIZE 2/  >NOT
 11    PAD WORKSET WORKSIZE 2/  >OR
 12    #STEPS @ 1+ #STEPS !   6 #STEPS @ QVERB !
 13    #STEPS @ QOBJ !   OPEN !   ;

SCR # 80
  0 CR ." SCREEN 80: QUERY LANGUAGE - 5 "
  1
  2 : FIRST#  ( -- )  ( reset current bit# to beginning )
  3    0  WORK#  !   ;
  4
  5 : NEXT#  ( -- )  ( return number of next 1 bit in WORKSET )
  6    OPEN @   DFILE @RCNT 1+   SWAP   OPEN !  ( set limit )
  7    WORK# @ 1+   2DUP > NOT   ( start 1 past last hit )
  8    IF  2DROP   0      ( no more?  return a zero )
  9    ELSE  >R 0 SWAP R>  ( set up zero return if loop expires )
 10          DO  WORKSET  I  ?BIT   ( check one bit )
 11              IF  DROP  I  DUP WORK# !  LEAVE  THEN
 12          LOOP
 13    THEN   ;

SCR # 81
  0 CR ." SCREEN 81: QUERY LANGUAGE - 6 "
  1
  2 : .COUNT  ( -- )  ( print count of working set )
  3    FIRST#   0      ( initialize bit# and count )
  4    BEGIN  NEXT#   WHILE  1+   REPEAT   .   ;
  5
  6 : .SET  ( -- )  ( list record #s in current set )
  7    FIRST#
  8    BEGIN  NEXT#   WHILE  WORK# @ .   REPEAT   ;

SCR # 82
  0 CR ." SCREEN 82: UTILITY WORDS "
  1
  2 : LOAD-DB  ( -- )  ( reset defaults & load user's def'ns )
  3    16 KWD/REC !   NONE;
  4    1 LOAD    DFILE   ;
  5 : USER-DB ;   ( dummy header to start user's def'ns )
```

Listing continued on next page

```
( The Indexer -- sample session )
ok
ok
FLUSH ok
LOAD-DB ok
ok
ok
( Since this is a new database, we have to enter ) ok
( the keyword terms we intend to use. ) ok
ok
KFILE ok
NEW-KWD
ENTER NEW KEYWORD: BICYCLING
BICYCLING                                ok
ok
NEW-KWD
ENTER NEW KEYWORD: BRAIN/MIND
BRAIN/MIND                               ok
ok
NEW-KWD
ENTER NEW KEYWORD: COMPUTERS
COMPUTERS                                ok
ok
NEW-KWD
ENTER NEW KEYWORD: NETWORKING
NETWORKING                               ok
ok
ok
( Now add some data records and index them ) ok
ok
DFILE ok
NEW-DATA
Enter contact name: JEFFREY SCHWINN
Enter telephone #: (805) 555-1212
Enter network address: TZA999
JEFFREY SCHWINN              (805) 555-1212   TZA999

ok
+TERMS BICYCLING,NETWORKING
BICYCLING
NETWORKINGok
ok
NEW-DATA
Enter contact name: MARK I. RIS
Enter telephone #: (619) 555-3730
Enter network address: 12345,765
MARK I. RIS                  (619) 555-3730   12345,765

ok
+TERMS COMPUTERS,BRAIN/MIND,NETWORKING
COMPUTERS
BRAIN/MIND
NETWORKINGok
ok
NEW-DATA
Enter contact name: SHERRIE JOLLYGOOD
Enter telephone #: (713) 555-9317
Enter network address:  (none)
SHERRIE JOLLYGOOD            (713) 555-9317    (none)
ok
+TERMS NETWORKING,COMPUTERS
NETWORKING
COMPUTERSok
ok
NEW-DATA
Enter contact name: MO TOBE CANE
Enter telephone #: (415) 555-1031
Enter network address: OMNI/1031/CZ
MO TOBE CANE                 (415) 555-1031   OMNI/1031/CZ

ok
+TERMS NETWORKING,BICYCLING
NETWORKING
BICYCLINGok
ok
( Now we can process the database for specific keywords ) ok
ok
SELECT( NETWORKING) ok
.COUNT 4 ok
ok
AND( BICYCLING) ok
.COUNT 2 ok
.SET 1 4 ok
ok
.QUERY
SELECT( NETWORKING)
AND( BICYCLING) ok
ok
ok
( Make a special purpose word to print "hits" ) ok
: TELLME   FIRST#  0 BEGIN NEXT# WHILE WORK# @ READ .REC REPEAT ; ok
ok
TELLME
JEFFREY SCHWINN              (805) 555-1212   TZA999

MO TOBE CANE                 (415) 555-1031   OMNI/1031/CZ

ok
ok
FLUSH ok
```

End of Listing

each command in FORTH is executed left to right, all operators are at equal precedence and are interpreted in the order given.

Multiple commands can be given on a line if right parentheses are used as delimiters between them. **SELECT(ARTS) AND(HABITATS)** is an example of a valid query.

The next two query words enable a user to process the working set. **FIRST#** sets the current bit number in **WORK#** back to zero. **NEXT#** returns the record number of the next member of the working set or zero if there are no more.

.COUNT and **.SET** are examples of using **FIRST#** and **NEXT#**. They interrogate the working set and for each member perform some action (**.COUNT** just increments a tally on the stack and prints the total when done; **.SET** prints the record number of each member.)

LOAD-DB resets some default values and starts loading the user's data base definition screens. **USER-DB** is another null word that can be used to **FORGET** a data base and load another without having to reload the Indexer.

The case study included with this article is an index of resource persons, their telephone numbers and computer network addresses. Starting with a newly formatted diskette, with all binary zeroes in each block, the definitions shown were typed onto screens 1-3 using a FORTH text editor. Then the Indexer was loaded.

In the sample session, a **FLUSH** is first done to end off processing the previous data base, if any. Then **LOAD-DB** resets some variables and loads the user's data base definitions. Next the keywords file is opened by the word **KFILE**, and **NEW-KWD** is used to enter the keywords that will be describing data records in this data base. This only needs to be done at the beginning or when additional terms need to be defined. Four terms are defined in the sample: **BICYCLING**, **BRAIN/MIND**, **COMPUTERS** and **NETWORKING**.

Data records are then added to the data base using the word **NEW-DATA**. After each record is entered, the keywords applicable to it are entered via **+TERMS**. In the sample, a total of four people are entered into the data base with a variety of interest keywords

associated with them. **-TERMS** could have been used to remove a keyword from a particular data record, to correct errors or reflect a change; **.TERMS** could have been used to check which keywords are defined for the current data record.

Once a data base exists, we can use the query language to extract a subset of it. In the example, we first choose a working set equal to all people involved in networking using the **SELECT(NETWORKING)** query. **.COUNT** tells us that this includes everyone. To narrow the search, we use **AND(** to require "hits" to also be into bicycling. **.COUNT** now tells us the working set consists of two members, and **.SET** lists their record numbers: 1 and 4.

At this point we can use **READ** to make one of these the current record, and process it in some manner; we decide instead to define a new word that will print the record image for all records in the working set. This we give the name **TELLME**, and when executed we get a listing of records 1 and 4. Finally, we do another **FLUSH** before removing our newly updated data base from the drive.

IMPLEMENTATION NOTE: This program was written using A-FORTH, a version which runs on Z80/8080 systems under Micropolis MDOS operating system. You should be aware that ' (tick) is immediate in A-FORTH, and that the block size on the diskette is 1024 bytes.

REFERENCES

1. Cramer, Robert F. "*Understanding Keywords for Creative Selective Dissemination of Information*" *Lateral Thinking* 6, February 1980, pp. 1-6.

2. Glass, Harvey. "Functional Programming and FORTH" *FORTH Dimensions*, January/February 1982, pp. 137-8.

3. Haydon, Glen B. "Elements of a FORTH Data Base Design" *FORTH Dimensions*, July/August 1981, pp. 45-52.

4. Reese, Peter. "A Disk Operating System for FORTH" *BYTE*, April 1982, pp. 322-85. □

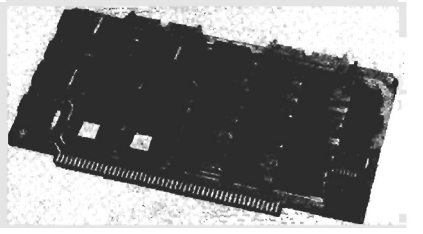
DEVELOPMENT TOOLS

Develop FORTH code for any target 8080/Z80 system on your current 8080/Z80 or Cromemco CDOS based system

8080/Z80 METAFORTH CROSS-COMPILER

- Produces code that may be downloaded to any Z80 or 8080 processor
- Includes 8080 and Z80 assemblers
- Can produce code without headers and link words for up to 30% space savings
- Can produce ROMable code
- 79 Standard FORTH
- Price $450

No downloading – No trial PROM burning. This port-addressed RAM on your S-100 host is the ROM of your target system



WORD/BYTE WIDE ROM SIMULATOR

- Simulates 16K bytes of memory (8K bytes for 2708 and 2758)
- Simulates 2708, 2758, 2516, 2716, 2532, 2732, 2564 and 2764 PROMS
- The simulated memory may be either byte or 16-bit word organized
- No S-100 memory is needed to hold ROM data
- Driver program verifies simulated PROM contents
- Price $495 each

CONSULTING SERVICES

Inner Access provides you with Custom Software Design. We have supplied many clients with both Systems and Application Software tailored to their specific needs. Contact us for your special programming requirements.

FORTH WORKSHOPS

ONE-WEEK WORKSHOPS — ENROLLMENT LIMITED TO 8 STUDENTS

FORTH Fundamentals

- Program Design
- Program Documentation
- FORTH Architecture
- FORTH Arithmetic
- Control Structures
- Input/Output
- The Vocabulary Mechanism
- Meta-Defining Words

OCT. 4-8 NOV. 8-12
JAN. 3-7 FEB. 7-11

$395 Incl. Text

Advanced FORTH Applications

- FORTH Tools
- Engineering Applications
- Floating Point
- Communications
- Sorting & Searching
- Project Accounting System
- Process Control
- Simulations

NOV. 15-19
FEB. 14-18

$495 Incl. Text

Advanced FORTH Systems

- FORTH Internals
- Assemblers and Editors
- Other Compilers
- Cross-Compilation Theory
- Romability, Multitasking, Timesharing
- File Systems/ Database Systems

OCT. 11-15
JAN. 10-14

$495 Incl. Text

Instructors: LEO BRODIE, GARY FEIERBACH and PAUL THOMAS

(For further information, please send for our complete FORTH Workshop Catalog.)

Inner Access Corporation

P.O. BOX 888 • BELMONT, CALIFORNIA 94002 • (415) 591-8295

A PICTURE Worth a 1000 Words

Elmer W. Fittery
International Computers
St. Louis, Missouri

This article describes a set of words which we use to create user-friendly video display screens for data input and output in business applications.

The general defining word **PICTURE** creates classes of data structures (e.g., alpha, 32-bit dollar amounts, etc.). The name of each of these classes is also, in turn, a defining word which creates individual fields of that data type, with a specified location on the screen and a specified text description. Let's look at some examples:

EXAMPLE #1 — Use **PICTURE** to build an input/output field for names of real estate companies you contact concerning buying beach front property. Assume names are less than 20 characters in length. Input/output is to occur on row 5, column 0.

SOLUTION —

```
     " A20"                PICTURE  20ALPHA
0 5  " COMPANY NAME: "     20ALPHA  [COMPANY.NAME]
```

The solution is a two step process. First we used the word **PICTURE** to build the word **20ALPHA**. The A in the string preceding **PICTURE** specifies the word built by **PICTURE** will build alpha/numeric-input/output fields. The 20 following the A specifies allow for input/output of 20 bytes. After building **20ALPHA** I build the actual input/output field **[COMPANY.NAME]**. Now let's use this mysterious word **[COMPANY.NAME]** I built with **20ALPHA**.

By typing **HEAD [COMPANY.NAME]** your cursor is positioned to column = 0, row = 5 and the field heading for the word **[COMPANY.NAME]** i.e. COMPANY NAME: is typed.

By typing **PUT [COMPANY.NAME]** your cursor is positioned to column = 0, row = 5 and the field heading for **[COMPANY.NAME]** i.e. COMPANY NAME: is typed. Subsequently text in **PAD** for 20 characters is typed.

By typing **GET [COMPANY.NAME]** the cursor is positioned to column = 0, row = 5 and the field heading for **[COMPANY.NAME]** is typed — 20 spaces are typed in inverse video following video — the cursor is positioned to the first position of the inverted video field. Input may then proceed until the inverted video field is filled. If you try to input too many characters a bell will sound. To specify you are finished with input press your return key. The return key is that key which responds with 13 when you execute **KEY**. If data is entered it is placed at **PAD** and a flag of 1 will be placed on the stack, otherwise a false flag of 0 will be placed on the stack.

By typing **PUT&GET [COMPANY.NAME]** a **PUT [COMPANY.NAME]** will be performed followed by a **GET [COMPANY.NAME]**. If the **GET [COMPANY.NAME]** returns a false flag, the **PUT [COMPANY.NAME]** will be performed again. NOTE: A false flag specifies no input was done, a true flag specifies input was done, and data is at **PAD**.

Usually the **PUT&GET** mode is used for displaying default information, then allowing the user to accept the default or enter his own information.

This was the first of three examples in using **PICTURE**. If you have a FORTH system running get the code **PICTURE** running and try the above example.

EXAMPLE #2 — Build an input/output field for cost of beach front property. This field is to display the cost in the following format. $Z,ZZZ,ZZ9.99 where Z specifies zero suppress, 9 specifies hard digit output and the dollar sign, comma and decimals mean outputing those characters when appropriate. This field is to start on row = 8, column = 0.

SOLUTION —

```
     " $Z,ZZZ,ZZ9.99"      PICTURE  2DOLLAR
0 8  " PROPERTY COST: "    2DOLLAR  [PROPERTY.COST]
```

The **HEAD**, **GET**, **PUT** and **PUT&GET** modes function exactly the same as when using them with an alpha/numeric input field. Data input/output is handled a little differently. First only digits 0-9 are allowed as input, second input is from right to left (NON-PROGRAMMERS are more familiar with calculator style input).

By typing **HEAD [PROPERTY.COST]** the cursor is positioned to row = 0, column = 8 — and the heading is typed out.

By typing **PUT [PROPERTY.COST]** the **HEAD** function is performed, the double number (32 bit value) is masked appropriately then displayed.

By typing **GET [PROPRTY.COST]** the **HEAD** function is performed, input from the field is commenced. Every time a key 0-9 is pressed, the new value is displayed properly formatted. The number of Z's and 9's used when building **2DOLLAR** specify the maximum number of digits to be input. Exceeding this limit will cause a bell to sound. When you press the return key, a true flag will be placed on the stack above the double, i.e., a 32 bit value if input was done, otherwise a false flag will be placed on the stack.

By typing **PUT&GET [PROPERTY.COST]** the **PUT** function is performed, followed by the **GET** function. If the **PUT&GET** causes a false flag to be placed on the stack the original value displayed by the **PUT** function will be redisplayed. In all other respects the **PUT&GET** function is the same as the **GET** function .

EXAMPLE #3 — Define an input/output field for area codes. It is to be of the form (999) where 9 specifies a hard digit. The field is to be on row = 10, starting at column = 35.

SOLUTION —

```
     " (999)"              PICTURE  16DIGIT
35 10 " AREA CODE: "       16DIGIT  [AREA.CODE]
```

[AREA.CODE] functions the same as **[PROPERTY.COST]** except it handles 16 bit values as opposed to 32 bit values. I want to get into some technical stuff now. When **PICTURE** builds a word the character string proceeding the

Continued

word **PICTURE** is examined to determine whether **PICTURE** is to build an alpha/numeric, double, or single input/output building word. In the case of alpha/numeric fields, the count for the input/output field is determined by looking at the character sequence following the A — in our first example it was 20. This specified a 20 byte alpha/numeric input/output field building word was to be created by **PICTURE**. If A25 was used in place of A20 in example #1 you would have built a 25 byte alpha/numeric input/output field building word. In the case of numeric input/output the digits allowed to be input is determined by the number of Z's and 9's, the output window size is defined to allow all digits and all mask characters to be displayed. The other problem I had to address was — does a word built by **PICTURE** build words handling 16 bit values or 32 bit values? My approach to this problem follows.

16 bit values may only display numbers from 0 to 9999, i.e., I do not handle negative numbers, and I do not try to process numbers larger than 10,000 with a 16 bit field handling word. 32 bit values may only display numbers from 0 to 9,999,999,999 — again no negative numbers. Not handling negative numbers and numbers above 9,999,999,999 is only a matter of what my applicaiton code needs are. If the total number of Z's and 9's is greater than 5, **PICTURE** will build a word which builds 32 bit input/output handling fields, otherwise it will build a word which builds 16 bit input/output handling fields.

Figure 1 shows an example of using my screen handling technique.

As you can see changing headings for fields, changing the position on the screen where input/output fields process data, changing masks used by fields is very simple. Simple screens may be built in 15 minutes, while very detailed screens may be built in approximately one hour.

One final note, when you set your mode it remains the same until you change it. I.e., in the word **DISPLAY-SCREEN, HEAD** is in effect when all field words are executed.

Feel free to use this code for personal use. Contact us concerning commercial use. Our hardware consists of a IBM Personal Computer, two Apples, three floppy drives and one 10meg Corvus drive. Our software development system supports:

1. Index files which allow access to any of 1,079,127 records by key in two disk reads. (Based on 6 byte key and a 16 byte record.)
2. Direct files which allow access to any of 2,147,483,647 records in one disk read.
3. Merge sort package, report generator binary overlays, head and link stripping vocabulary which stores stripped names and links on disk.

For more information contact Elmer W. Fittery at International Computers, 13048 Olive Blvd., St. Louis, MO 63141, (314) 878-3228. □

Figure 1

```
 " A2"                 PICTURE   2ALPHA
 " A20"                PICTURE  20ALPHA
 " A25"                PICTURE  25ALPHA
 " A30"                PICTURE  30ALPAH
 " A64"                PICTURE  64ALPHA
 " $Z,ZZZ,ZZ9.99"      PICTURE  DOLLAR.MASK
 " (999)"              PICTURE  AREA.MASK
 " 999-9999"           PICTURE  PHONE.MASK
 " 99999"              PICTURE  ZIP.MASK

00 02 " REALTOR: "    20ALPHA  [COMPANY.NAME]
00 04 " SALESMAN: "   30ALPHA  [SALESMAN.NAME]
00 06 " STREET: "     25ALPHA  [STREET.ADDRESS]
00 08 " CITY: "       20ALPHA  [CITY]
30 08 " STATE: "       2ALPHA  [STATE]
40 08 " ZIP CODE: "   ZIP.MASK  [ZIP.CODE]
00 10 " AREA CODE: "  AREA.MASK [AREA.CODE]
20 10 " PHONE# : "    PHONE.MASK [PHONE.NUMBER]
00 12 " PROPERTY DESCRIPTION: " 64ALPHA [DESCRIPTION]
00 14 " PROPERTY COST: "        DOLLAR.MASK [PROPERTY.COST]
 To display the full screen -

      : DISPLAY-SCREEN HEAD  [COMPANY.NAME] [SALESMAN.NAME]
           [STREET.ADDRESS] [CITY] [STATE] [ZIP.CODE]
           [AREA.CODE] [PHONE.NUMBER]
           [DESCRIPTION] [PROPERTY.COST]  ;
```

```
SCR # 130
  0 (  SCREEN GEN      1 OF 5      (092881 EWF)          (  032982 EWF)
  1
  2  0   CONSTANT CRT:FLAG
  3  0   CONSTANT CRT:ARRAY
  4
  5 : HEAD     0 ' CRT:FLAG ! ;
  6 : PUT      1 ' CRT:FLAG ! ;
  7 : GET      2 ' CRT:FLAG ! ;
  8 : PUT&GET  3 ' CRT:FLAG ! ;
  9 -->
 10
 11
 12
 13
 14
 15

FIND ->                                    INSERT ->


SCR # 131
  0                                                  (  032982 EWF)
  1
  2 : (HEAD:ADDR) CRT:ARRAY 4 + @ ;
  3 : (HEAD:CNT)  CRT:ARRAY 2 + @ ;
  4 : (HEADER) (HEAD:ADDR) (HEAD:CNT) ;
  5 : (HEAD:XY) CRT:ARRAY 8 + @
  6             CRT:ARRAY 6 + @ ;
  7 : (MASK:TYPE)  CRT:ARRAY @ 2+ @ ;
  8 : (MASK:ADDR)  CRT:ARRAY @ 6 + @ ;
  9 : (MASK:CNT)   CRT:ARRAY @ 4 + @ ;
 10 : (MASKER)     (MASK:ADDR) (MASK:CNT) ;
 11 : (DIGIT:CNT) CRT:ARRAY @ @ ;
 12 : (MASK:XY) (HEAD:XY) SWAP
 13      (HEAD:CNT) + SWAP ;
 14 -->
 15

FIND ->                                    INSERT ->


SCR # 132
  0 (  SCREEN GEN      2 OF 5      (092881 EWF)          (  032982 EWF)
  1
  2 : (MASK) CRT:ARRAY @ 8 + @ EXECUTE
  3   (MASK:CNT) OVER - SPACES ;
  4
  5 : PUT:HEADER ( HEADER ONLY )
  6     (HEAD:XY) XY (HEADER) TYPE
  7     (MASK:CNT) SPACES
  8     (MASK:XY) XY ;
  9
 10 : PUT:STRING
 11     PUT:HEADER PAD (MASK:CNT) TYPE ;
 12
 13 : PUT:MASKED (MASK:XY) XY
 14     (MASK) TYPE  ; -->
 15

FIND ->                                    INSERT ->


SCR # 133
  0                                                  (  032982 EWF)
  1 0 VARIABLE #CNT
  2 : ASK PB 1+ 0
  3   DO KEY DUP 13 =
  4      IF DROP I #CNT ! 500 ( LEAVE +LOOP )
  5      ELSE DUP 8 =
  6         IF R IF BS SPACE 32 PAD I 1- + C!
  7              ELSE DROP 7
  8              THEN R> 1- 0 MAX >R 0 SWAP
  9         ELSE R> R> 2DUP >R >R 1- =
 10            IF DROP 0 7
 11            ELSE DUP 1  ROT2  PAD I + C!
 12            THEN
 13         THEN EMIT
 14      THEN
 15    +LOOP ;  -->

FIND ->                                    INSERT ->
```

Listing continued on next page

PICTURE (continued)

```
SCR # 134
  0                                                        (  032982 EWF)
  1 : @STRING ASK #CNT @ 0 > ( 0 #CNT ! ) ;
  2
  3 : GET:STRING PUT:HEADER INVERT CRT:FLAG 2 =
  4     IF PB THEN PAD (MASK:CNT) TYPE
  5     (MASK:XY) XY (DIGIT:CNT) @STRING
  6     (MASK:XY) XY PAD (MASK:CNT) NORMAL TYPE ;
  7
  8 : PUT&GET:STRING PUT:STRING GET:STRING ;
  9
 10 -->
 11
 12
 13
 14
 15

FIND ->                                    INSERT ->


SCR # 135
  0 (  SCREEN GEN      3 OF 5      (092881 EWF)          (  032982 EWF)
  1 : @NUMBER  1+ 0 0. 2SWAP
  2     DO KEY 13 OVER =
  3        IF DROP I #CNT ! 500
  4        ELSE 48 57 ROT RANGE?
  5           IF R> R> 2DUP >R >R 1- =
  6              IF DROP 7 EMIT 0
  7              ELSE >R 10 D*S R> 48 - M+ 1
  8              THEN
  9           ELSE 8 =
 10              IF R> 1- 0 MAX >R 10 D/S
 11              THEN 0
 12           THEN 3 PICK 3 PICK (MASK:TYPE) 1 =
 13           IF DROP THEN PUT:MASKED
 14        THEN
 15      +LOOP  #CNT @ IF 1 ELSE 2DROP 0 THEN 0 #CNT ! ; -->

FIND ->                                    INSERT ->


SCR # 136
  0                                                        (  012982 EWF)
  1
  2 : PUT?GET:SGL PUT:HEADER INVERT
  3    CRT:FLAG 2 = IF 0 THEN DUP
  4    PUT:MASKED (DIGIT:CNT) @NUMBER
  5    IF DROP SWAP DROP 1 OVER
  6    ELSE 0 SWAP
  7    THEN NORMAL PUT PUT:MASKED ;
  8
  9  : PUT?GET:DBL PUT:HEADER INVERT
 10    CRT:FLAG 2 = IF .0 THEN 2DUP
 11    PUT:MASKED (DIGIT:CNT) @NUMBER
 12    IF 2SWAP 2DROP 1 3 PICK 3 PICK
 13    ELSE 0 ROT2
 14    THEN NORMAL PUT PUT:MASKED ; -->
 15

FIND ->                                    INSERT ->


SCR # 137
  0 (  SCREEN GEN      4 OF 5      (092881 EWF)          (  032982 EWF)
  1
  2 : PUT:NUMBER ( HEADER+PUT  )
  3     PUT:HEADER PUT:MASKED ;
  4
  5 : PUT?GET:NUMBER ( HEADER+PUT+GET)
  6    (MASK:TYPE) 2 =
  7    IF PUT?GET:DBL
  8    ELSE PUT?GET:SGL
  9    THEN ;
 10
 11 -->
 12
 13
 14
 15

FIND ->                                    INSERT ->
```

```
SCR # 138
  0                                                  (  012982 EWF)
  1
  2 : CRT:ACTION
  3     <BUILDS 4 0 DO , LOOP DOES> SWAP 2 *
  4     + @ EXECUTE ;
  5
  6 ' PUT?GET:NUMBER CFA    ' PUT?GET:NUMBER CFA
  7 ' PUT:NUMBER     CFA    ' PUT:HEADER     CFA
  8   CRT:ACTION NUMBER:ACTION
  9
 10 ' PUT&GET:STRING CFA        ' GET:STRING CFA
 11 ' PUT:STRING     CFA        ' PUT:HEADER CFA
 12   CRT:ACTION STRING:ACTION
 13
 14 -->


SCR # 139
  0 (  SCREEN GEN      5 OF 5      (092881 EWF)          (  032982 EWF)
  1
  2 : ACTION CRT:FLAG DUP >R (MASK:TYPE) 0=
  3   IF  STRING:ACTION
  4   ELSE  NUMBER:ACTION
  5   THEN R> ' CRT:FLAG ! ;
  6
  7 : GET-COUNT  0 #CNT ! OVER C@ 65 =
  8     IF 2DUP PE 1 -1 D+ S@ .NUM?
  9        IF DROP ELSE QUIT THEN
 10    ELSE 2DUP 1+ OVER + SWAP
 11       DO I C@ 90 =     I C@ 57 =     OR #CNT +!
 12       LOOP #CNT @
 13   THEN ;
 14
 15 : STRING:FLD? OVER C@ 65  = ;    -->


SCR # 140
  0 (  SCREEN GEN      5 OF 5      (092881 EWF)          (  032982 EWF)
  1 : GET-TYPE STRING:FLD?
  2     IF 0
  3     ELSE #CNT @  5 <
  4          IF 1 ELSE #CNT @ 10 < IF 2  ELSE  3 THEN THEN
  5     THEN ;
  6 : ACTUAL-KEEP
  7     , ( ADDRESS OF MASK/TYPE ARRAY )
  8     , ( ADDRESS OF HEADER          )
  9     , ( LENGTH OF HEADER           )
 10     , ( Y POSITION HEADER          )
 11     , ( X POSITION OF HEADER       ) ;
 12 : ACTUAL <BUILDS ACTUAL-KEEP DOES>
 13     ' CRT:ARRAY ! ACTION ;
 14 : $? (MASK:ADDR) C@ 36 = IF 36  HOLD  THEN ;
 15 -->


SCR # 141
  0                                                  (  032982 EWF)
  1 : MASK-BUILDER (MASK:TYPE) 2 MOD IF S->D THEN <#
  2         (MASKER) OVER + 1-         ( ADDRESS OF MASK  )
  3         DO I C@ 57 =
  4            IF # -1
  5            ELSE I C@ 90 =
  6               IF 2DUP DO= IF $? LEAVE ELSE # THEN -1
  7               ELSE  I 1+ C@ 90 =
  8                  IF 2DUP DO=
  9                     IF $? -1000
 10                     ELSE I C@ HOLD  # -2
 11                     THEN
 12                  ELSE I C@ HOLD -1
 13                  THEN
 14               THEN
 15            THEN          +LOOP #)     ;   -->


SCR # 142
  0                                                  (  032982 EWF)
  1
  2 : PICTURE-KEEP GET-COUNT DUP >R
  3                ,  ( GET NUMBER OF Z, 9, OR A )
  4       GET-TYPE  DUP ,   ( 0=ALPHA, 1=SINGLE, 2=DOUBLE, 3=TRIPLE)
  5       IF R> DROP ELSE DROP R> THEN
  6                 ,              ( MASK SIZE    )
  7                 ,              ( MASK ADDRESS )
  8       ' MASK-BUILDER  CFA ,    ( EXECUTE WORD ) ;
  9
 10 : PICTURE  <BUILDS PICTURE-KEEP DOES> ACTUAL ;
```

End of Listing

FORTH PROGRAMMING AIDS

from Curry Associates

FORTH PROGRAMMING AIDS is a software package containing high-level FORTH routines that allow you to write more efficient programs in less development time. It is also useful for maintaining existing FORTH programs. The ***FPA*** package includes four modules:

TRANSLATOR provides a one-to-one translation of FORTH run-time code.

CALLFINDER finds calling words, i.e. calls *to* a specific word.

DECOMPILER generates structured FORTH source code from RAM and inserts program control words (*e.g.,* IF, ELSE).

SUBROUTINE DECOMPILER finds called words, *i.e.,* words called *by* a specific word, to all nesting levels.

FORTH PROGRAMMING AIDS enables you to:

- Minimize memory requirements for target systems by finding only those words used in the target application.
- Tailor existing words (including nucleus words) to specific needs by decompiling the word to disk, editing, and recompiling.
- Build on previous work by extracting debugged FORTH routines (including constants and variables) from RAM to disk.
- Patch changes into existing compiled words in seconds.

FORTH PROGRAMMING AIDS comes with complete source code and a 50-page, indexed manual.

The DECOMPILER alone is worth a second look. This is a *true* decompiler which converts the FORTH words in RAM into compilable, structured FORTH source code, including program control words such as IF, ELSE, THEN, BEGIN, etc. If you ask ***FPA*** to DECOMPILE the nucleus word INTERPRET, you get the following output displayed on your terminal within 3 seconds:

```
                ( NFA&PFA: 4796 4810 )
: INTERPRET
    BEGIN -FIND
       IF STATE @ <
          IF CFA ,
          ELSE CFA EXECUTE
          THEN ?STACK
       ELSE HERE NUMBER DPL @ 1+
          IF [COMPILE] DLITERAL
          ELSE DROP [COMPILE] LITERAL
          THEN ?STACK
       THEN
    AGAIN ;
```

You can decompile one word, or a range of words at one time — even the whole FORTH system! This decompiled output may be sent by ***FPA*** options to the console, printer, or disk.

DECOMPILE is useful for looking up words, or for obtaining variations of words by decompiling to disk, editing, and recompiling.

System Requirements: FORTH nucleus based on the fig-FORTH model or 79-STANDARD; a minimum of 3K bytes and a recommended 13K bytes of free dictionary space.

For more information, call **Ren Curry 415/322-1463** or **Tom Wempe 408/378-2811**

Yes, send me a copy of ***FORTH PROGRAMMING AIDS***, including all source code and the 50-page manual.

☐ fig-FORTH model	$150	Calif. residents add 6.5% tax.
☐ FORTH-79 STANDARD (specify system)	$150	Foreign air shipments add $15.
☐ Manual alone (credit toward program purchase)	$25	
☐ Send more information		

☐ Master Charge ☐ Visa Account Number _____ Exp. Date _____

Name _____

Company _____

Street _____

City/State/Zip _____

Indicate disk format:

- ☐ 8" ss/sd fig-FORTH screens
- ☐ 8" ss/sd CP/M™ 2.2 file
- ☐ Apple 3.3
- ☐ PC FORTH
- ☐ Other _____

Send to: **Curry Associates, P. O. Box 11324, Palo Alto, CA 94306 415/322-1463 or 408/378-2811**

Parameterized CREATE DOES>

Henry Laxen

Last time I talked about names and the importance of choosing them carefully. We also looked at an example of how the **STACK** data structure can be handy, and that there is no requirement for naming stack operators with little arrows attached to them. This time we will see another example of where stacks are a useful data structure and the names we will choose for the stack operators again will not have little arrows attached to them. Also we'll tie in some of the **CREATE DOES>** concepts we learned in the previous three articles. If **CREATE DOES>** is still mysterious to you, I strongly suggest you reread the three-article series on defining words, and get comfortable with them.

Let's review defining words philosophically. One way of looking at them is that they are a mechanism for building small custom compilers. By using defining words, you can specify both the *compile* time and the run time behavior of subsequent words. This gives you incredible flexibility, but occasionally you want even more. That is where what I call "parameterized defining words" come in. Sometimes you may want to create a defining word that has a general structure that is always the same, but some of the details may change depending upon what happens. For example, suppose you had a defining word that defines words that look up something in a file, and do one of three things depending upon whether the data element was missing in the file, uniquely present in the file, or had multiple occurrences in the file. The basic structure of the defining word can take the data element, look it up by say, sequentially searching the file, and then do something depending on the data element and the result of the lookup. Conventional defining words fail for this kind of task, since the action to be taken must be specified in the **DOES>** portion of the defining word, and cannot really be dependent upon the actual data element that was searched for.

One way to solve this problem is to allow for some kind of parameters other than those passed on the stack, inside of defining words. That way a defining word becomes a template whose actions can be customized later depending upon your desires and requirements. The idea then is to leave room for some number of parameters in the defining word and then recall and execute them at runtime. In Fig. 1 you will see a screen of FORTH code that accomplishes this task. Let's examine it carefully.

The first word, called **PARAMETERS** allocates space in the parameter field portion of the word being defined, and leaves a pointer to the beginning of the parameter field on the stack. The next word, **C:** patches the parameter field that was left on the stack with the parameter field of a colon definition, and starts compiling. The **[' : CFA @] LITERAL** code lays down the code field address of colon. The **HERE OVER !** phrase patches the parameter field that was left on the stack, and finally the **2+** increments the parameter field to the next one to patch. One note of caution here, the FORTH that I am using is Starting FORTH compatible, and **EXECUTE** operates on parameter fields, not code fields. If you have a FIG or FORTH-79 system you should move the **HERE** word to the very beginning of the definition, before the **[**. You will then be patching in code fields rather than parameter fields. It's a little detail that will cause your system to crash if you leave it out, otherwise there are no serious consequences. Next **C;** terminates a code segment begun with **C:** and resumes interpretation.

The next three words are actually used inside the **DOES>** part of the **CREATE DOES>** word to set things up and perform them. **INSTALL** should be used right after the **DOES>** word to push the parameter field of the definition onto the **STACK**. **REMOVE** is used to pop the **STACK**, and should be included everywhere in the **DOES>** part where we exit from the definition. Finally **P** is used to execute a particular parameter from the currently installed parameterized **CREATE DOES>** word. Why, you may ask, did I go through the trouble of implementing a stack to perform these operations. The answer is that I wanted to be able to nest these structures, and simply using a variable instead of a stack would have caused instant death. By using a stack, words defined by parameterized defining words can call other words defined by parameterized defining words. The only rule that must be obeyed is that whatever is **INSTALL**ed must be **REMOVE**d before exiting the **DOES>** portion of a word.

```
 0 \ Parameterized CREATE DOES> Structures
 1 : PARAMETERS    (S n — addr )
 2    HERE SWAP 2* ALLOT    ;
 3 : C:    (S n — n' )                    (for Forth-79)
 4    [ ' : CFA @ ] LITERAL ,   HERE OVER !   2+    ]    ;
 5 : C;    (S — )
 6    COMPILE EXIT    [COMPILE] [    ;    IMMEDIATE
 7 CREATE STACK    HERE ,    10 ALLOT
 8 : INSTALL    (S n — )    ( Push onto the STACK )
 9    STACK    2 OVER +!    @ !    ;
10 : REMOVE    (S — )
11    -2 STACK +!    ;
12 : P    (S n — )
13    2*    STACK @ @    + @ EXECUTE    ;
14
15
```

Figure 1

Now let's take a look at Fig. 2 for an example of how to use the beasts we have just constructed. We define a simple yet flexible case statement that allows us to use code fragments rather than just single words for the different cases. The compile time part of the definition simply allocates some space in the parameter field of the word being defined, and leaves the address of the beginning of the parameter field on the stack. Notice that it reads very nicely. If you knew how many parameters you wanted (some constant number) in the **CREATE** part of a defining word, you could take care of them by simply saying **CREATE 5 PARAMETERS** . . . for 5. In the case of **CASES**, we will specify the number of parameters by placing a number on the stack before invoking **CASES**. The runtime part simply **INSTALL**s the pfa of the word defined onto the **STACK** we created, executes the phrase corresponding to the number on the stack with **P**, and **REMOVE**s the pfa from the **STACK**. The code in lines 4 thru 8 illustrate a word which prints out the first nine Roman numerals based upon what is on the stack. Notice that we did not have to create a name for each of the 10 cases, we simply compiled the code for each case with a **C:** and **C;** pair, which patched their runtime address into the appropriate slot in the parameter field of **ROMAN**. The **DROP** on line 8 is needed to remove the incremented parameter field of **ROMAN** which is still on the stack at this point. We could have given it the more impressive name of something like **END-CASE** by defining **: END-CASE DROP ;** .

Now let's take a look at a slightly more non-trivial example in Fig. 3. Here we define a word called **INFORM** which prints a message about a number both in Arabic notation and in some other notation which we will specify later. In fact parameter 0 of the definition is in charge of printing out the name of the format in which the number is being displayed. Parameter 0 is executed by the little code fragment **0 P** on line 5. Thus if we were to execute the line **8 ENGLISH** we would get:

The number is 8 in Arabic, and
Eight in English text

```
 0 \ Example of use for parameterized CREATE DOES>
 1 : CASES    ( A generalized CASE statement )
 2    CREATE (S n — )    PARAMETERS
 3    DOES>  (S n — )    INSTALL P REMOVE   ;
 4 10 CASES   ROMAN        ( 0 ) C: ." ???" C;
 5    ( 1 ) C: ." I" C;    ( 2 ) C: ." II" C;    ( 3 ) C: ." III" C;
 6    ( 4 ) C: ." IV" C;   ( 5 ) C: ." V" C;     ( 6 ) C: ." VI" C;
 7    ( 7 ) C: ." VII" C;  ( 8 ) C: ." VIII" C;  ( 9 ) C: ." IX" C;
 8 DROP
 9
10
11
12
13
14
15
```

Figure 2

```
 0 \ Example of use for parameterized CREATE DOES>
 1 : INFORM    ( A little more substantial example )
 2    CREATE (S n — )    PARAMETERS
 3    DOES>  (S n — )    INSTALL
 4       CR ." The number is " DUP . ." in Arabic, and "
 5       P ( number ) ."  in " 0 P ( type )   REMOVE   ;
 6 10 INFORM  ROMAN        ( 0 ) C: ." Roman numerals" C;
 7    ( 1 ) C: ." I" C;    ( 2 ) C: ." II" C;    ( 3 ) C: ." III" C;
 8    ( 4 ) C: ." IV" C;   ( 5 ) C: ." V" C;     ( 6 ) C: ." VI" C;
 9    ( 7 ) C: ." VII" C;  ( 8 ) C: ." VIII" C;  ( 9 ) C: ." IX" C;
10    DROP
11 10 INFORM  ENGLISH    ( 0) C: ." English text" C;
12 ( 1) C: ." One" C;    ( 2) C: ." Two" C;    ( 3) C: ." Three" C;
13 ( 4) C: ." Four" C;   ( 5) C: ." Five" C;   ( 6) C: ." Six" C;
14 ( 7) C: ." Seven" C;  ( 8) C: ." Eight" C;  ( 9) C: ." Nine" C;
15    DROP
```

Figure 3

Notice how the defining word **INFORM** mixes both constant functions, such as printing out the carriage return, the message about Arabic, and the actual Arabic number, with the variable data of the number in some other format, and the name of the format. This concept of parameterizing your defining words can lead to substantial code savings, and considerable simplifications in many cases where you want to do almost the same thing many times, but a few of the little details are different. I find that the most common use for this concept is in error handling, since what is being manipulated by the defining word is always the same, but you often want to do something special and data dependent if an error occurs. You can use parameterized defining words to help you out in those situations.

Next time we'll explore the how and why of compiling words, and look at some non-obvious examples of how to use them. We can use compiling words to change the syntax of FORTH, and create special compiling structures in ways totally different from defining words. Until then good luck, and may the FORTH be with you.

Henry Laxen in an independent FORTH consultant. This is the first anniversary of his Techniques Tutorial column, and we personally think it is an excellent series. Thank you, Henry, for your continuing contributions. —Editor □

Summary of Changes in Proposed FORTH-83

Robert L. Smith

The FORTH Standards Team met in Carmel Valley on October 3-5, 1982. A number of changes were made to Draft A (the working draft of the proposed FORTH-83 Standard. The team has instructed me to publish a brief description of the technical proposals passed at the meeting. This summary is given below. Due to lack of time and space, I will defer more extensive comments for later issues of *FORTH Dimensions*. Team sponsors have received most of the proposals listed below, along with others for which no action was taken, or which were rejected by the team. The following proposals were accepted by the team:

Prop.#	Author	Description
39	Tenney	Modify definition of BUFFER, and move to the Controlled Reference Word Set.
204	Referees	The serial number of a glossary definition will indicate the year in which a substantive change was made.
205	Referees	Expand definition of DOES> so that the creation of a new word may use CREATE or any user-defined word which executes CREATE.
207	Referees	Append an M to glossary definitions to indicate those words which may have multi-programming implications. Referee clarification.
208	Referees	SAVE-BUFFERS to Required Word Set. FLUSH executes SAVE-BUFFERS then unassigns the buffers. EMPTY-BUFFERS is moved to Controlled Reference Word Set.
209	Berkey, et al	The true value returned by Standard words has all bits set to 1.
210	Smith	Add the Immediate attribute to LEAVE.
213	Tenney	KEY does not display the character, but EXPECT does. Clarification of Standard action of EMIT.
219	Shaw	PAD holds at least 84 bytes. Other clarifications.
222	Shaw	Clarification of error conditions for D.R. and U.R. Referees to consider.
223	Thomas	Define concept of a FORTH word. Referee clarificaiton.
224	Thomas	Clarify definition of Compilation Address. Referees to consider.
225	Thomas	Clarify definition of "compilation." Referees to consider.
226	Thomas	Clarify definition of "glossary." Referees to consider.
227	Thomas	Clarify definition of "interpreter." Referees to consider.
228	Thomas	Additional definitions for clarity. Referee clarification.
249	Schleisiek	Use 0 based arguments for PICK and ROLL.
256	Wasson	Move QUERY to the Controlled Reference Word Set.
265	Oakley	Clarifications for "." # LOAD CONVERT EXECUTE and BLK. For referee consideration.
283	Sanderson	Add a new word LEAVES to the Controlled Reference Word Set.
284	Rarthe & Sanderson	Add SPAN to Required Word Set. Span returns the address of a word holding the number of characters actually received by EXPECT.
292	Tenney	Change meaning of NOT to yield the one's complement of the input argument.
296	Perry & Shaw	Move LIST and SCR to the Controlled Reference Word Set.
297	Perry & Shaw	Delete ? from the Required Word Set.
299	Ragsdale	Move CURRENT and CONTEXT to the System Word Set. Modify text of VOCABULARY, :, and FORGET.
301	MOD Subteam	The integer quotient returned from signed division functions is the floor of the real number quotient. The MOD value has the sign of the denominator.
304	MOD Subteam	Name changes: U* to UM* and U/MOD to UM/MOD.
306	Schleisiek	In EXPECT, one or two nulls might be stored at the end of the text.
308	Forsley	Add the word RECURSE to the Controlled Reference Word Set.
309A	James	FIND, ', and ['] return the compilation address. EXECUTE takes a compilation address.

Continued

CHANGES (continued)

309B	James	The word BODY converts a compilation address to a parameter field address.
310	Patten & James	FIND takes a string address and returns a compilation address and a flag indicating whether the word was found, and if so, whether the word is immediate.
311	Harris	A Programmer's Bill of Rights.
313	Tenney	Labelling requirement for Environmental Dependencies.
314	Tenney	New wording for KEY and EMIT to allow environmental dependencies. Referees may clarify.
315	Currie	Clarification of DO-loop nesting levels.
317	McNeil	Return Stack Restrictions.
——	String Subteam	Maximum string lengths are 255 for ."", .(, and ABORT".
——	——	Delete last phrase in the definition of COMPILE.
——	String Subteam	Clarification of WORD.
——	——	Referees to define limitations of PAD use.
——	——	Referees to clarify USER variables.
——	——	Referees to resolve discrepancies of : and ; .
——	——	Referees to correct wording of CONVERT.
——	——	Referees given discretion of placement of words in layers.
——	——	Referees to give text on warning on the use of TIB.

The team has requested Bill Ragsdale to resubmit proposals 274 and 275 in proper form to be accepted as Experimental Proposals. Proposal 274 suggests that vocabularies be non-immediate. Proposal 275 deals with the "ONLY" concept for search orders. Proposal 216 was tabled, awaiting action by the Subteam on Strings, headed by Don Colburn. This proposal deals with in-line arguments.

The following proposals were considered, but failed to pass by the required 2/3 of the voting members present: 213A, 213D, 215, 255, 274, 279, 283-1, 289, 290, 295, 297, 298, 303. Two unnumbered proposals also failed to pass. One suggested that PAD be moved to the Reference Word Set, and the other would have changed UM/MOD to divide a 31 bit number by a 15 bit number.

I will have more information next time.

Robert Smith is the current Secretary of the FORTH Standards Team and was a member of the original FORTH Implementation Team for FIG. He is employed by ESL Inc. in Sunnyvale, California. □

: High-performance M68000
HighResGraphics polyFORTH/32++;

That's right. The power of polyFORTH/32 has been combined with the Omnibyte 68000 based system and high resolution graphics from Ikier Technology. You get big system performance at micro prices.

The standard system includes a 10-Mhz Motorola 68000 with the FORTH operating system. The base system also has 128K bytes of RAM expandable to 16M bytes. With a single interface board you can connect to disks, a Centronics printer, a realtime clock, and 48 lines of parallel I/O. Need file storage? The standard Omnibyte system has a 1M byte floppy, and a 20M byte hard disk. You have this and more.

High resolution graphics makes the picture complete. The Ikier HRG Display Controller gives you hardware vectors, arcs, pan, scroll, and zoom. There are two megapixels of display memory on a single board with a built in DMA function for image transfer. For color, multiple Ikier boards can be configured to give up to 256 colors from a palette of 2^{24} with two overlay planes. Graphics goes FORTH.

It's the realtime thing, polyFORTH/32. At your disposal you have a full 32 bit system with multitasking, an unlimited number of partions, 32 bit wide stacks, and realtime functions. Data base support includes direct, random, sequential, and indexed files. For graphics, there are easy to use graphic commands, and you can software define the CRT you are using.

The best news about our system is the price. To find out more and get a quotation call Randy Cochran, Omnibyte marketing manager, at 312-231-6880. Or if your interest is just graphics, call Larre Nelson, Ikier Technology marketing manager, at 617-275-4330.

NEW PRODUCT ANNOUNCEMENTS

FORTH ON SANYO

The Software Works announces the release of *Software Works FORTH, Level I*™ for the MBC-1000, 2000, and 3000 computers of Sanyo Business Systems Corporation. Software Works FORTH is a fully documented, well supported, enhanced, 79 Standard version of the FORTH programming language. Level I, a basic system that resembles fig-FORTH, retails for $95.00.

Software Works FORTH uses CP/M for file handling and disk input/output for sharing data files and disk storage with other languages and application programs.

Features include: a fig-style line editor; a full block editor; a versatile, modular assembler; a generalized interface for character I/O devices; a stack practice aid; and a collection of utilities for copying and deleting FORTH screens; terminal independent input and output; simultaneous output to several devices; and volume spanning.

Level II, a more comprehensive version of Software Works FORTH, is also available for $179.

The Software Works • 1032 Elwell Court, Suite 210 • Palo Alto, CA 94303 • (415) 960-1800

TRS 80 COLOR-FORTH

COLOR-FORTH for the TRS-80 Color Computer (6809) is now available from Hoyt Stearns Electronics. Prices are $58.95 for the RAM version and $123.00 for the ROM version + the RAM version. Included with the ROM are instructions for installing it in the disk controller, or it will work in a ROM cartridge.

COLOR-FORTH works with both disk and tape. The ROM version will work with 4K of RAM; the RAM based version requires 16K RAM.

COLOR-FORTH comes with a powerful Semigraphic-8 editor and a set of utilities. It has a unique trace feature, and handles interrupts cleanly in high level FORTH. COLOR-FORTH maintains the CPU carry flag, for easy implementation of extended math, and has words for bit manipulation, graphics, sound, task multiplexing, fast math, auto-repeat keys, control keys, disk and tape control, and linkage to Basic routines.

COLOR-FORTH was written largely in assembly language, specifically for the Color Computer. It takes advantage of the unique features of the Color Computer, and is very fast. COLOR-FORTH does not affect normal operation of the Color Computer.

Hoyt Stearns Electronics • 4131 E. Cannon Drive • Phoenix, AZ 85028 • (602) 996-1717

6502 FORTH ON CODOS

fig-FORTH upgraded to FORTH-79 with Double-Number Standard Extensions, for any 6502 based Microcomputer running Micro Technology Unlimited's CODOS Disk Operating System. Changes within the nucleus give a 20-30% increase in execution speed over original fig-FORTH 6502.

The system interfaces to floppy disk via CODOS SVCs, and allows for up to eight simultaneously active files, as well as more than 800 screens per disk (when using double sided disks), and direct to disk data transfer capability. fig-FORTH 79 comes complete with an advanced Editor, a 6502 Assembler, many useful utility screens and a comprehensive User Manual. It is supplied on an 8 inch single sided double density disk or on a cassette at a price of $145.00 for orders from inside the USA, and without the disk interface to MTU's CODOS, for any 6502 based Microcomputer that can read either KIM or MOS Technology Standard recording formats, at a price of $38.00. For those who don't mind the typing, the listing for the complete system is available at a price of $34.00.

Mark I. Manning • 7611 Autumnal Lane • Liverpool, NY 13088

FORTH-32

Quest Research has released FORTH-32, a complete software development system for the IBM PC.

FORTH-32 is a language system which allows the programmer to easily utilize all of the IBM PC one megabyte memory domain. It does this by allowing a 32-bit addressing mode (as the FORTH-32 name implies) as well as a 16-bit addressing mode. In effect, FORTH-32 makes the segment structure of the memory management transparent to the user; thus, the programmer has direct access to all of the available memory without the need of specifying a segment address.

Addresses in the first 64K bytes relative to the load address can be specified with a single word (16 bits). Only when a program spans more than 64K bytes is 32-bit addressing needed. A MODE SENSING switch determines whether 16 bits are sufficient to specify an address, or if 32-bits are necessary. At run time, the interpreter properly unravels which mode was compiled into the dictionary and handles the addresses appropriately. In addition to this intermixed 16/32 bit addressing, most dictionary entries in the FORTH-32 model have two verbs tied to them. The first verb is a 32-bit version and resembles the standard fig-FORTH verb. The second verb, if present, is a 16-bit version. For example, there is a 32-bit and a 16-bit DO-LOOP present in the FORTH-32 model.

The FORTH-32 language is verb compatible with the fig-FORTH model. FORTH-32 is by nature a 32-bit language which can perform 16, 32, and 64 bit mathematical operations allowing extended precision arithmetic.

A utility allows one to transform a user-developed program into a marketable software package without requiring a licensing agreement with Quest Research, Inc. (as long as the entire FORTH-32 language is not included with the user software package). This is accomplished by building on disk a condensed executable file consisting of only those FORTH words needed for the software package.

The FORTH-32 Development System priced at $100 includes: Package Builder Utility, Assembler, Decompiler, Full Screen Editor, Debug (an extended FORTH debug with high level trace capability and full stack display), DOS interface (allows direct control of the disk sectors full file capability), Graphics (dot, line, circle, paint and color capability), Case Statement and User Manual. A Floating Point Package is also available for $50.

Quest Research Inc. • P.O. Box 2553 • Huntsville, AL 35804 • (800) 558-8088 • (205) 533-9405

FORTH/370 FOR LARGE IBM COMPUTERS

FORTH/370 is a fig-FORTH adaptation for IBM 370, 43X1, 303X, 308X, and equivalent computers. It runs under VM/CMS or MVS/TSO to give an office mainframe the power and flexibility of FORTH with program compatibility to micro fig-FORTH systems.

FORTH/370 is complete with a Reference Manual, Editor, Assembler, and a Library of Utilities. FORTH/370 disk files are compatible with and can be accessed by the host operating system. For example, in the CMS version either the CMS editor or the FORTH editor can be called from FORTH/370 to edit screens. Single and double numbers are 32 and 64 bits respectively.

FORTH/370 is available on 1600 or 6250 BPI tape for the introductory price of $1,350.00. Specify CMS or TSO. Maryland firms add 5% sales tax.

Ward Systems Groups • 8013 Meadowview Drive • Frederick, MD 21701

CURRY ASSOCIATES ANNOUNCES SOURCE INDEX

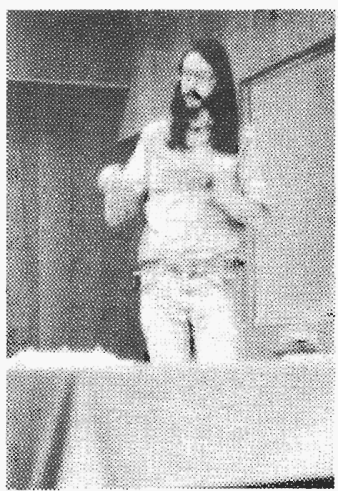
SOURCE INDEX is a software routine to aid in the development of large FORTH programs. Written in high level FORTH, it interprets the source code and collects the names of newly defined FORTH words. The program output consists of an alphabetical list of word names, the defining word (e.g., : VARIABLE), the word's location in the source code (screen and line number), and a count of defined words. An unusual feature of this routine is the ability to detect a new defining word (e.g., ARRAY) and the objects of this defining word (e.g., ARR1 ARR2).

Source Index includes all source code, manual, and program disk postpaid for $50. Specify fig-FORTH or FORTH-79 in one of the following disk formats: 8″ CP/M, 5″ PC-DOS, 5″ Apple 3.3.

Ren Curry • Curry Associates • P.O. Box 11324 • Palo Alto, CA 94306 • (415) 322-1463

4th FORML CONFERENCE
FORTH Modification Laboratory
October 6-7, 1982
Asilomar, California

Conference attendees (left to right) Michael Perry, Charles H. Moore, Glen B. Haydon and Michael Stolowitz break FORTH into laughter during a special Asilomar Session devoted exclusively to silliness.



Klaus Schleisiek delivers his presentation on "WORD Without a Reserved Character."

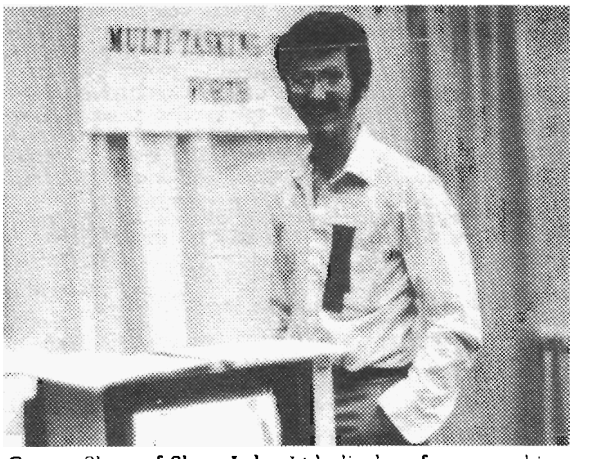
William Ragsdale explains his proposed vocabulary managment scheme (called ONLY) to Michael Stolowitz. The poster sessions provide an opportunity for one-on-one discussions of topics presented in the oral presentations.

Jill and Dick Miller present a poster session on turtle graphics.

Ed Rotberg plays his rendition of Ravel's "Bolero," which he programmed in FORTH on his Atari 800, for an appreciative Klaxon Suralis.

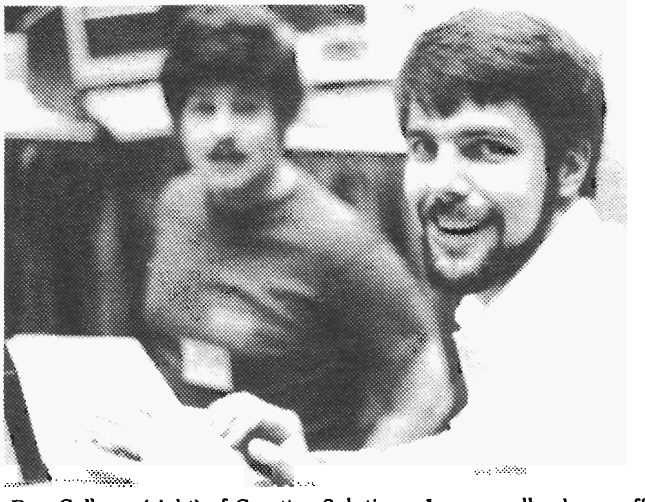
THE 4th ANNUAL FORTH NATIONAL CONVENTION
October 9, 1982
The Red Lion • San Jose, California

Dr. C.H. Ting (left) demos his new FORTH system to a prospect.

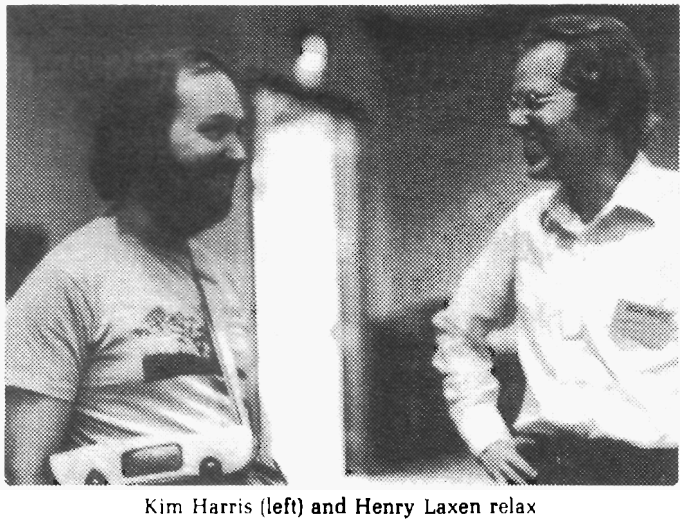


George Shaw of Shaw Labs, Ltd. displays fancy graphics running under his own multi-tasking FORTH.

John James, developer of the software for Communitree, explains how he implemented that computer conferencing program in the "Communications" panel discussion.



Don Colburn (right) of Creative Solutions, Inc. proudly shows off his multi-tasking FORTH for the Hewlett Packard 9826.



Kim Harris (left) and Henry Laxen relax during a break between presentations.

-=-=-=-=-

-=-=-=- 8080/Z80 FIG-FORTH for CP/M & CDOS systems -=-=-=-

-=-=-=-=-

$50 saves you keying the FIG FORTH model and many published FIG FORTH screens onto diskette and debugging them. You receive TWO 8 inch diskettes (single sided, single density, soft sectored only). The first disk is readable by Digital Research CP/M or Cromemco CDOS and contains 8080 source I keyed from the published listings of the FORTH INTEREST GROUP (FIG) plus a translated, enhanced version in ZILOG Z80 mnemonics. This disk also contains executable FORTH.COM files for Z80 & 8080 processors.

-=-=-=-=-

The 2nd disk contains FORTH readable screens including a extensive FULL-SCREEN EDITOR plus many items published in FORTH DIMENSIONS, including a FORTH TRACE utility, a model data base handler, an 8080 ASSEMBLER and formatted memory dump and I/O port dump words. The disks are packaged in a ring binder along with a complete listing of the FULL-SCREEN EDITOR and the FIG-FORTH INSTALLATION MANUAL (the language model of FIG-FORTH, a complete glossary, memory map, installation instructions and the FIG line editor listing and instructions).

-=-=-=-=-

This entire work is placed in the public domain in the manner and spirit of the work upon which it is based. Copies may be distributed when proper notices are included.

-=-=-=-=-

		USA	Foreign AIR
☐	Above described package	$50	$60
☐	Printed Z80 Assembly listing w/ xref..... (Zilog mnemonics)	$15	$18
☐	Printed 8080 Assembly listing.....	$15	$18
	TOTAL	$_____	_____

-=-=-=-=-

Price includes postage. No purchase orders without check. Arizona residents add sales tax. Make check or money order in US Funds on US bank, payable to:

Dennis Wilson c/o
Aristotelian Logicians
2631 East Pinchot Avenue
Phoenix, AZ 85016
(602) 956-7678

-=-=-=-=-

FIG Chapter News

Potomoc Chapter

At the October 5th meeting, Dennis Evans discussed "FORTH Applications in Optics." He described and demonstrated optic system design programs used at Goddard Space Flight Center. Dennis uses Microspeed, a FORTH system which includes a hardware floating point board, on an Apple computer.

At the November 2nd meeting the subject was "Commercial Programming with FORTH." Tom Arnall described his experience using FORTH to develop commercial software. He compared FORTH with other business software development languages. Tom recently completed an inventory system written in FORTH on a PDP-11.

Southern Ontario Chapter

At the December 4th meeting Dr. Charles E. Eaker discussed "Varieties of FORTH." Dr. Eaker is well known in the FORTH community for his version of the CASE statement, as well as for his implementation of CCForth on the TRS-80 Colour Computer. Dr. Eaker discussed his current project implementing a multi-tasking FORTH for 6809 systems running under the Flex operating system. His talk proved to be of interest to both beginners and experts alike, as it gave us an opportunity to meet with someone who has been involved with FORTH for a considerable time.

Anyone wishing to talk at future meetings please contact Dr. Solntseff in Hamilton, Ontario Canada at (416) 525-9140, ext. 2065.

FORTH Interest Group

At the October 28th meeting of the FORTH Interest Group in Phoenix, Arizona, Dennis Wilson, president of Aristotelian Logicians demonstrated his "Full Screen Editor for Disk and Memory." The Editor is written completely in fig-FORTH and was running on a Cromemco 3102 terminal. The Editor combines the ability to view and update memory instead of resorting to dumps, peeks and pokes, with full screen editing.

Upcoming Issues

Vol. V. No. 2
Project Management

Vol. V, No. 3
Guest Regional Issue

Vol. V, No. 4
FORTH Environment/Tools 8/1/83

Vol. V, No. 5
FORTH in the Laboratory 10/1/83

Vol. V, No. 6
Looking Back (History) 12/1/83

• The Institute for Applied Forth Research, Inc.

PURPOSE

- Purpose

In the past decade the Forth programming language has emerged as a powerful tool for applying computers. However, we feel a lack of application-oriented information has reduced its general acceptance. Recognizing this, we have chartered a not-for-profit corporation to support and promote Forth and its applications.

PROGRAMS

- Programs

The Institute will sponsor Forth-related conferences such as the 1981 and 1982 conferences at the University of Rochester. Other institutions using Forth may consider hosting conferences or seminars with administrative backing from the Institute. We will address specific topics in Forth through seminars, workshops, and lectures like those held this past May in Rochester.

We are starting a program of summer student fellowships at universities working with Forth, in which the projects will be chosen so as to further a student's expertise in Forth, while doing work of interest to the host institution and others. Further, in recognition of the fact that many students trained in the sciences have not had a chance to apply that science within the constraints of industry, we are arranging summer internships within companies applying Forth. Combination academic/industrial internships are also possible, and would facilitate the transfer of new techniques between the academic and industrial sectors. We welcome suggestions for this program, as well as inquiries from companies and institutions interested in sponsoring students.

Finally, we are establishing a library or archive of Forth-related materials to serve as a resource for the community.

PUBLICATIONS

- Publications

We plan to publish a refereed, professional journal, whose primary subjects will be Forth-based tools and their applications in industry and research. Referees are being chosen, based on their experience and interest, from universities, research laboratories, and businesses using Forth. We intend to publish papers not only by professional Forth programmers, but also by people who have used Forth as a tool to facilitate their own work. The journal will try to represent the growing Forth community, and provide a forum for original work.

The journal will appear twice the first year, and quarterly thereafter. The first issue will come out in January 1983.

The Institute will also undertake publication of the proceedings of the conferences it sponsors. The 1982 Rochester Forth Conference Proceedings should be available in September from Mountain View Press.

- If you are interested in helping to further the Forth concept through a unique organization, please contact:

Thea Martin, Executive Director
The Institute for Applied Forth Research, Inc.
70 Elmwood Avenue
Rochester, New York 14611
(716) 235-0168

Fig Chapters

U.S.

• ARIZONA

Phoenix Chapter
Dennis L. Wilson, Samaritan Health Services, 2121 E. Magnolia, Phoenix, AZ. 602/257-6875

• CALIFORNIA

Los Angeles Chapter
Monthly, 4th Sat., 11 a.m., Allstate Savings, 8800 So. Sepulveda Blvd., L.A. Philip Wasson 213/649-1428

Northern California Chapter
Monthly, 4th Sat., 1 p.m., FORML Workshop at 10 a.m. Palo Alto area. Contact FIG Hotline 415/962-8653

Orange County Chapter
Monthly, 4th Wed., 12 noon, Fullerton Savings, 18020 Brookhurst, Fountain Valley. 714/523-4202

San Diego Chapter
Weekly, Thurs., 12 noon. Call Guy Kelly, 714 268-3100 x4784

• MASSACHUSETTS

Boston Chapter
Monthly, 1st Wed., 7 p.m. Mitre Corp. Cafeteria. Bedford, MA. Bob Demrow. 617 688-5661 after 5 p.m.

• MICHIGAN

Detroit Chapter
Call Dean Vieau. 313 493-5105

• MINNESOTA

MNFIG Chapter
Monthly, 1st Mon. Call Mark Abbot (days) 612 854-8776 or Fred Olson, 612/588-9532. or write to: MNFIG, 1156 Lincoln Ave., St. Paul, MN 55105

• MISSOURI

St. Louis Chapter
Call David Doudna, 314/867-4482

• NEVADA

Las Vegas Chapter
Suite 900, 101 Convention Center Dr. Las Vegas, NV 89109, 702/737-5670

• NEW JERSEY

New Jersey Chapter
Call George Lyons. 201 451-2905 eves.

• NEW YORK

New York Chapter
Call Tom Jung. 212 746-4062

• OKLAHOMA

Tulsa Chapter
Monthly, 3rd Tues., 7:30 p.m., The Computer Store, 4343 So. Peoria, Tulsa, OK. Call Bob Giles, 918/599-9304 or Art Gorski, 918/743-0113

• OHIO

Dayton Chapter
Monthly, 2nd Tues., Datalink Computer Center, 4920 Airway Road, Dayton, OH 45431. Call Gary Ganger, (513) 849-1483.

• OREGON

Portland Chapter
Call Timothy Huang, 9529 Northeast Gertz Circle, Portland, OR 97211, 503/289-9135

• PENNSYLVANIA

Philadelphia Chapter
Call Barry Greebel, Continental Data Systems, 1 Bala Plaza, Suite 212, Bala Cynwid, PA 19004

• TEXAS

Austin Chapter
Call John Hastings, 512/327-5864

Dallas/Ft. Worth Chapter
Monthly, 4th Thurs. 7 p.m., Software Automation, 1005 Business Parkway, Richardson, TX. Call Marvin Elder, 214/231-9142 or Bill Drissel, 214/264-9680

• UTAH

Salt Lake City Chapter
Call Bill Haygood, 801/942-8000

• VERMONT

ACE Fig Chapter
Monthly, 4th Thur., 7:30 p.m., The Isley Library, 3rd Floor Meeting Rm., Main St., Middlebury, VT 05753. Contact Hal Clark, RD #1 Box 810, Starksboro, VT 05487, 802/877-2911 days; 802/453-4442 eves.

• VIRGINIA

Potomac Chapter
Monthly, 1st Tues. 7p.m., Lee Center, Lee Highway at Lexington Street, Arlington, Virginia. Call Joel Shprentz, 703/437-9218 eves.

• WASHINGTON

Seattle Chapter
Call Chuck Pliske or Dwight Vandenburg, 206/542-7611

FOREIGN

• AUSTRALIA

Australia Chapter
Contact Lance Collins, 65 Martin Rd., Glen Iris, Victoria 3146, or phone (03) 292600

• CANADA

Southern Ontario Chapter
Contact Dr. N. Solntseff, Unit for Computer Science, McMaster University, Hamilton, Ontario L8S 4K1, 416/525-9140 x2065

Quebec Chapter
Call Gilles Paillard, 418/871-1960 or 643-2561

• ENGLAND

English Chapter
Write to FORTH Interest Group, 38 Worsley Rd., Frimley, Camberley, Surrey, GU16 5AU, England

• JAPAN

Japanese Chapter
Contact Masa Tasaki, Baba-Bldg. 8F, 3-23-8 Nishi-Shimbashi, Minato-ku, Tokyo, 105 Japan

• NETHERLANDS

HCC-FORTH Interest Group Chapter
Contact F.J. Meijer, Digicos, Aart V.D. Neerweg 31, Ouderkerk A.D. Amstel, The Netherlands

• WEST GERMANY

West German Chapter
Contact Wolf Gervert, Roter Hahn 29, D-2 Hamburg 72, West Germany, (040) 644-3985

SPECIAL GROUPS

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

Detroit Atari FORTH
Monthly, 1st Wed.
Call Tom Chrapkiewicz
313/524-2100 or 313/772-8291

Nova Group Chapter
Contact Mr. Francis Saint, 2218 Lulu, Witchita, KS 67211, 316/261-6280 (days)

MMSFORTH Users Chapter
Monthly, 3rd Wed., 7 p.m., Cochituate, MA. Dick Miller, 617/653-6136

FORTH INTEREST GROUP

MAIL ORDER

	USA	FOREIGN AIR
☐ Membership in FORTH Interest Group and Volume V of FORTH DIMENSIONS	$15	$27
☐ Back Volumes of FORTH DIMENSIONS. **Price per each.** ☐ I ☐ II ☐ III ☐ IV	$15	$18
☐ fig-FORTH Installation Manual, containing the language model of fig-FORTH, a complete glossary, memory map and installation instructions	$15	$18
☐ Assembly Language Source Listings of fig-FORTH for specific CPU's and machines. The above manual is required for installation. Check appropriate box(es). **Price per each.** ☐ 1802 ☐ 6502 ☐ 6800 ☐ 6809 ☐ VAX ☐ 8080 ☐ 8086/8088 ☐ 9900 ☐ APPLE II ☐ ECLIPSE ☐ PACE ☐ NOVA ☐ PDP-11 ☐ 68000 ☐ ALPHA MICRO	$15	$18
☐ "Starting FORTH" by Brodie. BEST book on FORTH. (Paperback)	$16	$20
☐ "Starting FORTH" by Brodie. (Hard Cover)	$20	$25
☐ PROCEEDINGS 1980 FORML (FORTH Modification Lab) Conference	$25	$35
☐ PROCEEDINGS 1981 FORML Conference, Both Volumes	$40	$55
☐ Volume I, Language Structure	$25	$35
☐ Volume II, Systems and Applications	$25	$35
☐ PROCEEDINGS 1982 FORML Conference	$25	$35
☐ PROCEEDINGS 1981 FORTH Univ. of Rochester Conference	$25	$35
☐ PROCEEDINGS 1982 FORTH Univ. of Rochester Conference	$25	$35
☐ FORTH-79 Standard, a publication of the FORTH Standards Team	$15	$18
☐ Kitt Peak Primer, by Stevens. An in-depth self-study primer.	$25	$35
☐ BYTE Magazine Reprints of FORTH articles, 8/80 to 4/81	$ 5	$10
☐ FIG T-shirts: ☐ Small ☐ Medium ☐ Large ☐ X-Large	$10	$12
☐ Poster, August 1980 BYTE cover, 16" x 22"	$ 3	$ 5
☐ FORTH Programmer's Reference Card. If ordered separately, send a stamped, addressed envelope.		FREE

TOTAL $_____

NAME_____ MAIL STOP/APT_____

ORGANIZATION_____ PHONE ( )_____

ADDRESS_____

CITY_____ STATE_____ ZIP_____ COUNTRY_____

VISA #_____ MASTERCARD #_____

Expiration Date_____ (Minimum of $15.00 on charge cards)

Make check or money order in US Funds on US bank, payable to: **FIG.** All prices include postage. **No purchase orders without check.** California residents add sales tax. 1/83

ORDER PHONE NUMBER: (415) 962-8653

FORTH INTEREST GROUP * PO BOX 1105 * SAN CARLOS, CA 94070

FORTH INTEREST GROUP
P.O. Box 1105
San Carlos, CA 94070

BULK RATE
U.S. POSTAGE
PAID
Permit No. 2xx
Mtn. View, CA

940865ITHRRRRRP S
R L SMITH
TSL SUBSIDIAR OF TRW
PO BOX 510
SUNNYVALE, CA 94086