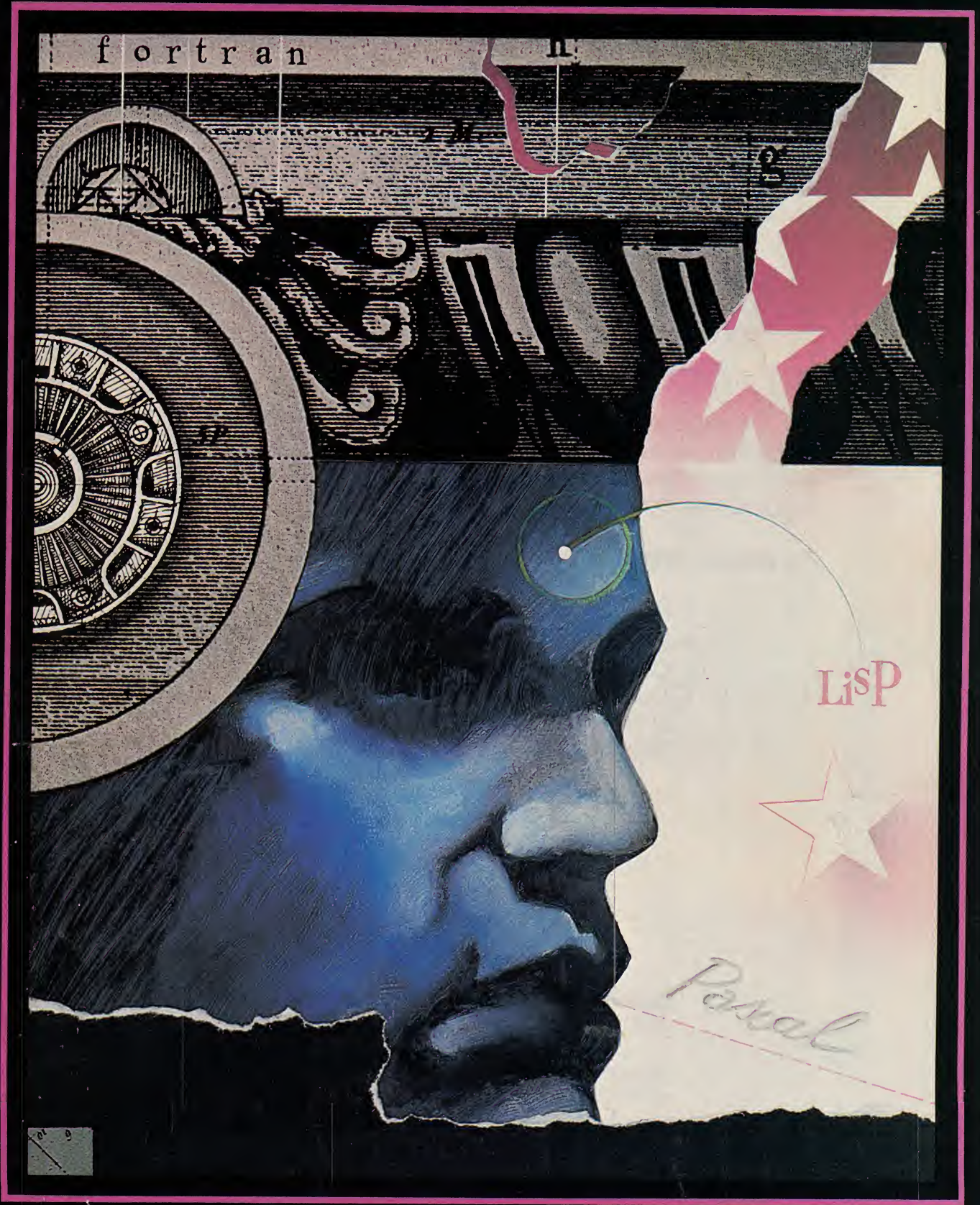


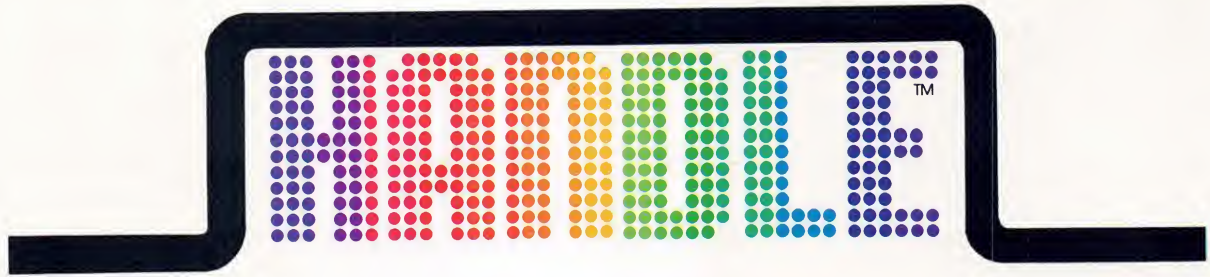
UNIX™ REVIEW

THE PUBLICATION FOR THE UNIX™ COMMUNITY

SEPTEMBER 1985 \$3.95



LANGUAGES



Modular. Integrated. Now.

Handle Writer/Spell™

Word processing with integrated spelling correction and verification.

Handle Calc™

Spreadsheet with up to 32,000 rows and columns. Conditional and iterative recalculation.

The **Handle Office-Automation Series** is a powerful set of modular, integrated software tools developed for today's multiuser office environment. **Handle** application modules can be used stand-alone or combined into a fully integrated system.

The **Handle Office-Automation Series** modules offer:

- Ease of Use and Learning
- Insulation from **UNIX**
- Data Sharing Between Multiple Users
- Data Integration Between Modules
- Data Sharing with Other Software Products
- Sophisticated Document Security System

Handle Technologies, Inc.

Corporate Office

6300 Richmond
3rd Floor
Houston, TX 77057
(713) 266-1415

Sales and Product Information

850 North Lake Tahoe Blvd.
P.O. Box 1913
Tahoe City, CA 95730
(916) 583-7283

How to go from UNIX to DOS without compromising your standards.

It's easy. Just get an industry standard file access method that works on both.

C-ISAM™ from RDS.

It's been the UNIX™ standard for years (used in more UNIX languages and programs than any other access method), and it's fast becoming the standard for DOS.

Why?

Because of the way it works. Its B+ Tree indexing structure offers unlimited indexes. There's also automatic or manual record locking and optional transaction audit trails. Plus index compression to save disk space and cut access times.

How can we be so sure C-ISAM works so well? We use it ourselves. It's a part of INFORMIX®, INFORMIX-SQL and File-it!™, our best selling database management programs.

For an information packet, call (415) 424-1300. Or write RDS, 2471 East Bayshore Road, Palo Alto, CA 94303.

You'll see why anything less than C-ISAM is just a compromise.



© 1985, Relational Database Systems, Inc. UNIX is a trademark of AT&T Bell Laboratories. INFORMIX is a registered trademark and RDS, C-ISAM and File-It! are trademarks of Relational Database Systems, Inc.

RELATIONAL DATABASE SYSTEMS, INC.

How we improved Structured Query Language.

Actually, we didn't change a thing.

We just combined it with the best relational database management system.

Introducing INFORMIX[®]SQL.

It runs on either UNIX[™] or MS[™]-DOS operating systems. And now with IBM's SQL

as part of the program, you can ask more of your database. Using the emerging industry-standard query language.

To make your job easier, INFORMIX-SQL comes with the most complete set of application building tools. Including a full report

writer and screen generator. Plus a family of companion products that all work together.

Like our embedded SQLs for C and COBOL. So you can easily link your programs with ours. File-it![™], our easy-to-use

file manager. And C-ISAM™, the de facto standard ISAM for UNIX. It's built into all our products, but you can buy it separately.

And when you choose RDS, you'll be in the company of some other good companies. Computer manufacturers including AT&T, Northern Telecom, Altos and over 60 others. And major corporations like Anheuser Busch, First National Bank of Chicago and Pacific Bell.

Which makes sense. After all, only RDS offers a family of products that work so well together. As well as with so many industry standards.

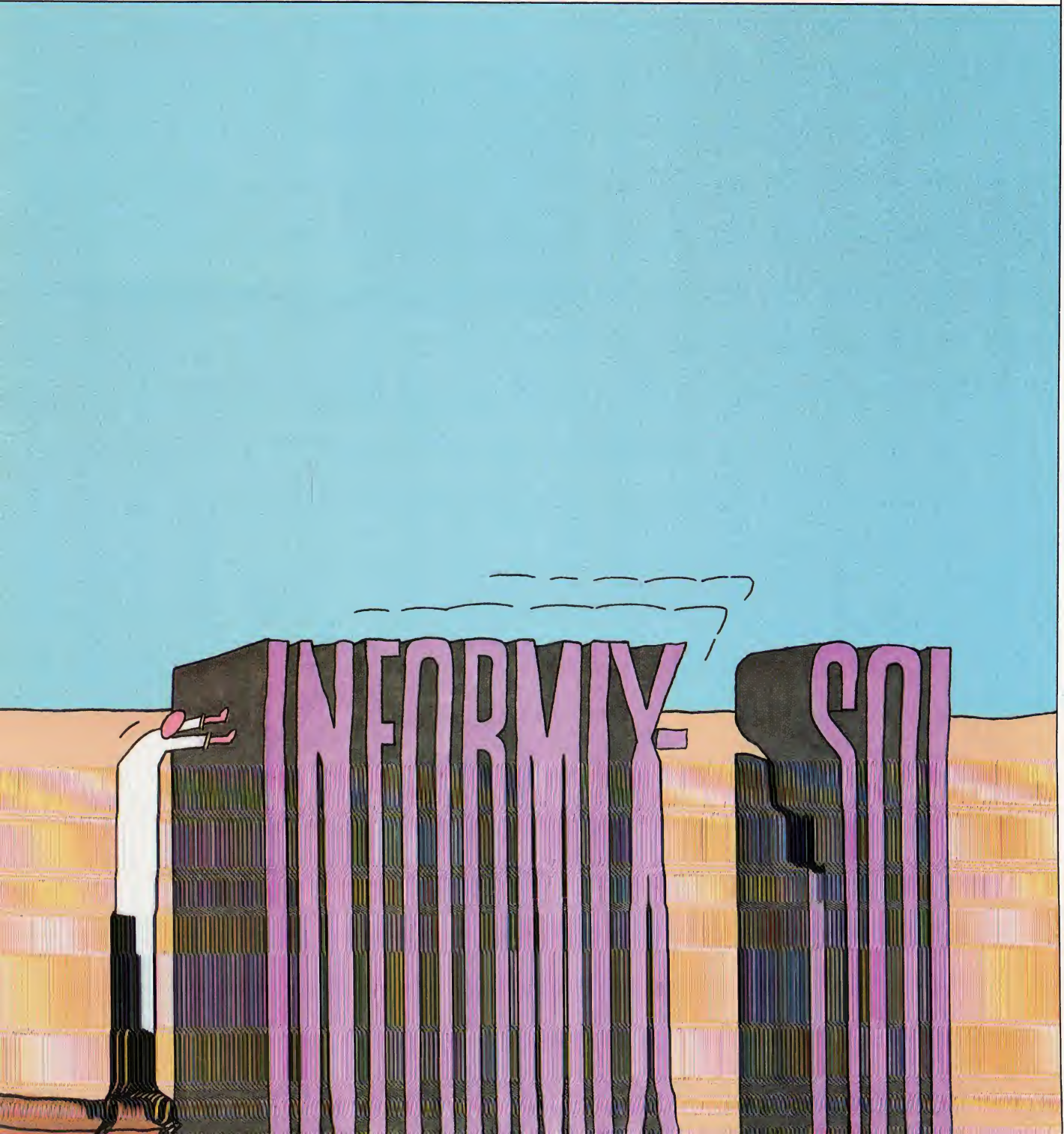
So call us for a demo, a manual and a copy of our Independent Software Vendor Catalog. Software vendors be sure to ask about our new "Hooks" software integration program. Our number: 415/424-1300.

Or write RDS, 2471 East Bayshore Road, Palo Alto, CA 94303.

And we'll show you how we took a good idea and made it better.



RELATIONAL DATABASE SYSTEMS, INC.





UNIXTM REVIEW

THE PUBLICATION FOR THE UNIX COMMUNITY

Volume 3,

Number 9

September 1985

DEPARTMENTS:

- 6 Viewpoint**
- 8 The Monthly Report**
By David Chandler
- 18 The Human Factor**
By Richard Morin
- 64 Industry Insider**
By Mark G. Sobell
- 68 C Advisor**
By Bill Tuthill
- 74 Rules of the Game**
By Glenn Groenewold
- 80 Devil's Advocate**
By Stan Kelly-Bootle
- 82 Problem Solver**
By Bob Toxen
- 88 The UNIX Glossary**
By Steve Rosenthal
- 92 Recent Releases**
- 100 Calendar**
- 104 The Last Word**
- 108 Advertiser's Index**

FEATURES:

22 THE RIGHT LANGUAGE FOR THE JOB

By Joel McCormack

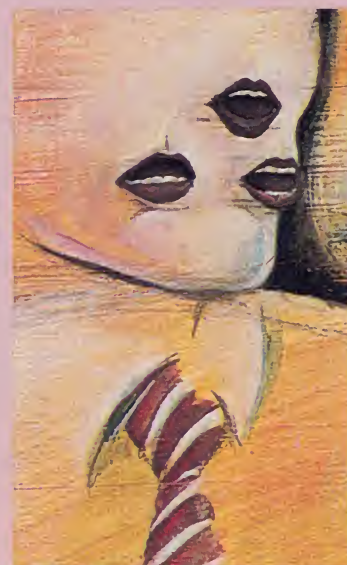
A survey of the principal languages available under UNIX.



35 THE HOUSE OF MANY TONGUES

By Steve Johnson

In the UNIX environment, there is a language to every purpose.



Cover art by Stephen G. Luker



LANGUAGES

38 SOURCE CODE MAINTENANCE

By Marc Rochkind

Where the challenges lie and how SCCS can help.



42 INTERVIEW WITH STU FELDMAN

By Dick Karpinski

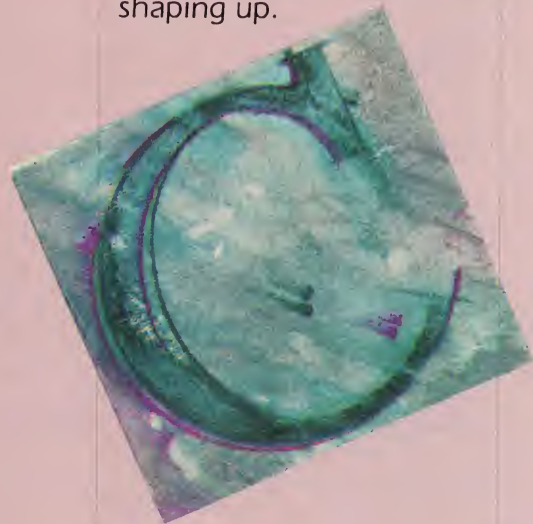
The author of **f77**, **EFL**, and **make** discusses the state of language technology.



50 TOWARDS ANSI C

By Thomas Plum

How the C standards efforts are shaping up.



51 LISP ON THE MOVE

By Joel Hass

What are the ties that bind UNIX and Lisp?

UNIX REVIEW (ISSN-0742-3136) is published monthly by REVIEW Publications Co. It is a publication dedicated exclusively to the needs of the UNIX community. Second class postage paid at Renton, WA 98055 and at additional mailing offices. POSTMASTER: Please send Form 3579 to UNIX REVIEW, 500 Howard Street, San Francisco, CA 94105. Entire contents copyright 1985. All rights reserved and nothing may be reproduced in whole or in part without prior written permission from UNIX REVIEW.

Subscriptions to UNIX REVIEW are available at the following annual rates (12 issues): US\$28 in the US; US\$35 in Canada; US\$48 in all other countries/surface mail; US\$85 in all other countries/airmail. Correspondence regarding editorial (press releases, product announcements) and circulation (subscriptions, fulfillment, change of address) should be sent to 500 Howard Street, San Francisco, CA 94105. Telephone 415/397-1881. Correspondence regarding dealer sales should be sent to 901 South 3rd Street, Renton, WA 98055. Telephone 206/271-9605.

Letters to UNIX REVIEW or its editors become the property of the magazine and are assumed intended for publication and may so be used. They should include the writer's full name, address and home telephone number. Letters may be edited for the purpose of clarity or space. Opinions expressed by the authors are not necessarily those of UNIX REVIEW.

UNIX is a trademark of Bell Laboratories, Inc. UNIX REVIEW is not affiliated with Bell Laboratories.

PUBLISHER:

Pamela J. McKee

ASSOCIATE PUBLISHERS:

Ken Roberts, Scott Robin

EDITORIAL DIRECTOR:

Stephen J. Schneiderman

EDITOR:

Mark Compton

ASSOCIATE EDITOR:

David Chandler

EDITORIAL ADVISOR:

Dr. Stephen R. Bourne, Consulting Software Engineer, Digital Equipment Corporation.

EDITORIAL REVIEW BOARD:

Dr. Greg Chesson, Chief Scientist, Silicon Graphics, Inc.

Larry Crume, Director, AT&T UNIX Systems Far East

Ted Dolotta, Senior Vice President of Technology, Interactive Systems Corporation

Gene Dronek, Director of Software, Aim Technology

Ian Johnstone, Project Manager, Operating Software, Sequent Computer Systems

Bob Marsh, Chairman, Plexus Computers

John Mashey, Manager, Operating Systems, MIPS Computer Systems

Robert Mitze, Department Head, UNIX Computing System Development, AT&T Bell Labs

Deborah Scherrer, Computer Scientist, Mt. Xinu

Jeff Schriebman, President, UniSoft Systems

Rob Warnock, Consultant

Otis Wilson, Manager, Software Sales and Marketing, AT&T Information Systems

HARDWARE REVIEW BOARD:

Gene Dronek, Director of Software, Aim Technology

Doug Merritt, Technical Staff, International Technical Seminars, Inc.

Richard Morin, Consultant, Santa Forda Computer Laboratory

Mark G. Sobell, Consultant

SOFTWARE REVIEW BOARD:

Ken Arnold, Consultant, UC Berkeley

Jordan Mattson, Programmer, UC Santa Cruz

Dr. Kirk McKusick, Research Computer Scientist, UC Berkeley

Doug Merritt, Technical Staff, International Technical Seminars, Inc.

Mark G. Sobell, Consultant

CONTRIBUTING EDITOR:

Ned Peirce, Systems Analyst, AT&T Information Systems

PRODUCTION DIRECTOR:

Nancy Jorgensen

PRODUCTION STAFF:

Cynthia Grant, Tamara V. Heimarck, Carrie

Hunkapiller, Florence O'Brien, Barbara Perry, Denise Wertzler

BUSINESS MANAGER:

Ron King

CIRCULATION DIRECTOR:

Wini D. Ragus

CIRCULATION MANAGER:

Jerry M. Okabe

MARKETING MANAGER:

Donald A. Pazour

OFFICE MANAGER:

Tracey J. McKee

TRAFFIC:

James A. O'Brien, Manager

Tom Burrill, Dan McKee, Corey Nelson

NATIONAL SALES OFFICES:

500 Howard St.

San Francisco, CA 94105

[415] 397-1881

Regional Sales Manager:

Colleen M. Y. Rodgers

Sales/Marketing Assistant:

Anmarie Achacoso

370 Lexington Ave.

New York, NY 10017

[212] 683-9294

Regional Sales Manager:

Katie A. McGoldrick

BPA membership applied for in March, 1985

VIEWPOINT

The way, the truth, and the life

I've often heard that technology is the religion of our age. Two years in the UNIX community have convinced me.

Independent of whatever feelings people on the outside might have, a sense of righteousness runs strong among the believers. Dutifully, many of the faithful have gone forth into the world and multiplied.

Alas, as with every religion, elders in this true Church seem ever to be embroiled in debate. Should services be conducted in C or Modula-2? Was not Lisp handed down on Mt. Sinai from the heavens?

Answers do not come quickly. As Steve Johnson notes, unto every purpose there is a language in the UNIX environment. It is neither necessary nor desirable that these be distilled into a single tongue, but natural selection nonetheless will take its course (a mixed metaphor if ever there was one).

Which languages will ascend to the altar and which will fall by the wayside? Debates on this question seem like so much crying in the wilderness.

Verily I say to you, all the preaching in the world cannot deter the mysterious ways in which inertia works. Once a language (or an operating system, for that matter) has become entrenched, momentum alone ensures that it will not easily be displaced. Would-be heirs must

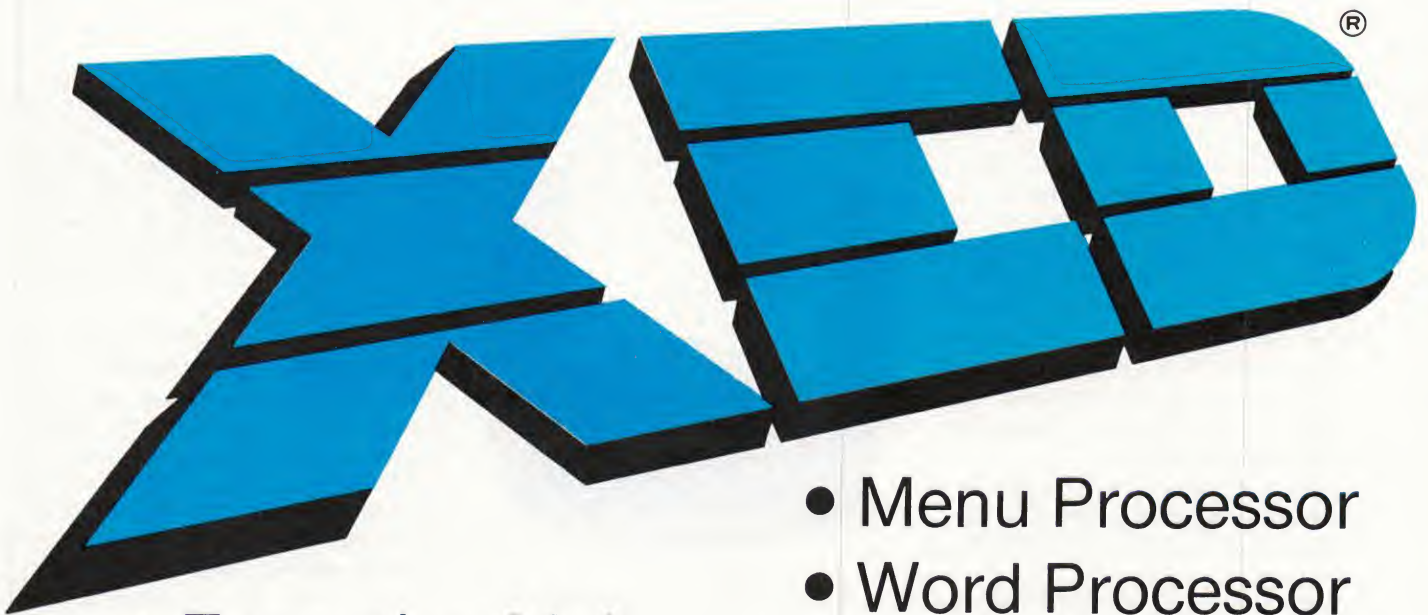
offer *significant* improvements to make inroads—particularly if investment in the incumbent has been substantial. Perhaps natural selection is predestined after all.

Make no mistake—C is the rock upon which UNIX is built. Although Joel McCormack makes a convincing argument in this month's lead article for the adoption of Modula-2 as the language of choice for UNIX development, he acknowledges that his cause faces an uphill fight. As evidence, he comments on the other major languages under UNIX, telling of the advantages they offer and the reasons they have failed to supplant C in the inner sanctum.

Modula-2 doubtless will grow in popularity, but C's defenders will not stand by idly. Never mind that C is getting on in years—that its design, though well suited to limited environments, has failed to keep pace with technology. The stake in C is great. Vast numbers of programmers have grown comfortable with it and the body of C software is substantial. The question, in any event, is strictly a religious matter.

Mark Compton

The First Name In Integrated Office Automation Software



- Executive Mail
- Telephone Directory

- Menu Processor
- Word Processor
- Forms/Data Base
- Spreadsheet

***Certified and
Deliverable Since 1981***

XED was the first independent software company to introduce a Unix WP package and achieved early success by selling to the government and international market (XED is the only Unix WP package to meet government specifications). Worldwide sales of XED rank Computer Methods first in both sales and units installed in 1984.



INTEGRATED OFFICE SOFTWARE

Box 3938 • Chatsworth, CA 91313 U.S.A. • (818) 884-2000
FAX (818) 884-3870 • Intl. TLX 292 662 XED UR

XED is a registered trademark of CCL Datentechnik AG
UNIX is a trademark of AT & T Bell Laboratories, Inc.

Circle No. 279 on Inquiry Card

THE MONTHLY REPORT

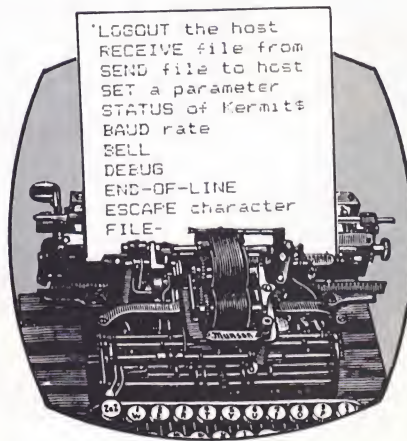
AT&T: Guns are the bread and butter

by David Chandler

Andrew Pollack of *The New York Times*, commenting on a recent announcement made by AT&T, referred to what he called the company's "drowsy start in the computer business". Some days previous, *The Wall Street Journal* had run a page-two story on the same announcement, relating it to AT&T's "fledgling computer business".

Before loyalists to the telecommunications giant start generalizing about the eastern establishment media, though, pause to consider the adjectives "drowsy" and "fledgling". The terms themselves don't indicate failure, nor inability—"drowsy", of course, describes non-energetic activity, and "fledgling" refers to a young bird just "fledged", having just acquired the feathers necessary for flight. The point is that the potential for vigor and success is present but not yet exhibited. After all, learning to fly is a process, and AT&T only now is learning to flap its wings.

The announcement to which the *Times* and *Journal* were referring called attention to a contract awarded to AT&T by the US Department of Defense (DoD) to provide, install, and maintain System V-based, 3B Series computer systems. Should the DoD exercise all options, the contract's value would approach \$946 million, the largest contract of its kind for AT&T.



"This is a very large procurement which we worked very hard on for more than a year," said Warren Corgan, Vice President of AT&T Federal Systems Division. Indeed, the DoD selected AT&T over many major vendors, including IBM, DEC, Honeywell, Sperry, and Gould. Neil Yelsey, an analyst with Solomon Brothers Inc., was quoted by the *Journal*, "This is a shot in the arm [this is, remember, a defense contract] for both AT&T's architecture and its (UNIX) operating system".

For all the work, money, and attention the contract brings, both AT&T and the government were very limited in their comments, declining to elaborate on a one-page press release "cleared through proper channels". On the one hand this is understandable, Defense Department regulations being what they are. A source inside AT&T even ad-

mitted that the *Journal* article was "not in keeping with DoD policy". The contract is actually with the National Security Agency, an intelligence organization within the DoD.

On the other hand, there is ever-present speculation on how such deals as this one will affect computer industry standards. Billion-dollar contracts for machines running UNIX raise eyebrows, especially when one of the parties is the federal government, a strong voice in the setting of standards. For the time being, however, given the restrictions on government comment, the UNIX community is limited to speculation. Voices proclaiming UNIX success generally can offer public knowledge, but when they enter the DoD cloister, they must take a vow of silence.

The DoD project will be managed by AT&T's Federal Systems Division based in Greensboro, NC. A field office will be set up in the Washington, DC, area to provide on-site support.

WE WILL SAY THIS

A topic AT&T willingly addresses is its June announcement of some 70 hardware and software products. Of significance among these are: 1) two new machines based on the WE32100 processor, the 3B2/400 and the 3B15; 2) communication processors for connecting

*“Our customers told
us what they wanted
in word processing:*

Compatibility

Flexibility

Ease of Learning

Vendor Reliability

Cost Effectiveness



*We listened.
The Professional Writer's
Package is the result.”*

Ed Zucker, UNIX Specialist
Emerging Technology

Compatibility. A word processor for all machines, UNIX or MS-DOS™.

Flexibility. A word processor for every use, whether it's writing complicated manuals or preparing simple letters.

Ease of Learning. Time and money shouldn't be wasted in training. On-line help, on-line tutorials with step-by-step instructions.

Vendor Reliability. Extensive word processing experience, a large installed base, accessible technical support, and a liberal update policy.

Cost Effectiveness. With these capabilities you can't afford not to have the Professional Writer's Package.™

EMERGING
TECHNOLOGY

Call or write for information:
4760 Walnut Street, Boulder, Colorado 80301 800/782-4896 303/447-9495

Circle No. 253 on Inquiry Card

UNIX is a trademark of Bell Laboratories. MS-DOS is a trademark of Microsoft Corporation. Professional Writer's Package is a trademark of Emerging Technology Consultants, Inc.

3Bs to mainframes; and 3) System V-VM.

The 3B2/400 is a System V.2-based desktop computer for use in the 10-25 user range, with standard features including the 32100 processor (32/32-bit architecture), 1 MB of RAM, a 10 MHz clock, a 720 KB floppy disk drive, one or two integral hard disk drives (each 30 MB or 72 MB), and an integral 23 MB cartridge tape backup unit. Options are emphasized, including the ability to expand RAM to 4 MB, and a Math Accelerator Unit (MAU), also known as a 32106 co-processor. Four configurations are suggested, ranging in price from \$19,950 to \$36,550.

The 3B15 is an upgrade of the 3B5, with the same architecture and CPU from the same family as for the smaller machine, but also offering a demand-paged management system, 40 percent more CPU power, and mandatory file and record locking. The 3B15's 32100 processor runs at 14 MHz under System V Release 2.1 (2.1 denoting the support of demand paging and a more powerful C compiler), with 2 MB of RAM, the 32106 MAU, and an 8 KB cache memory. Three suggested configurations for the 3B15 are priced from \$54,500 to \$64,500.

While enhancements to the 3B series are welcome, a point not immediately clear is which specific customer markets AT&T is targeting with these upgrades. Company representatives claim that, depending on the software, the 3B2/400 and 3B15 can be used either in scientific or office automation applications. But these days it is asking a great deal of a machine to succeed in either of these environments, and to expect it to succeed in both is an excessively heavy burden. While AT&T mentions the needs of Fortune 2000 firms and the 3B enhancements that were made to

meet these needs, the 400 and the 15 also have or will offer such things as the MAU and demand-paging—features usually associated with scientific applications.

The connection of the AT&T Communication Processor to Fortune 2000 applications is more readily apparent. A hardware interface "intended primarily to

**Voices proclaiming
UNIX success generally
can offer public
knowledge, but when
they enter the DoD
cloister, they must take
a vow of silence.**

serve the mainframe connectivity needs of users of networked 3B computers using AT&T 3BNET", the processor is a node on a 3BNET network that links 3Bs to IBM hosts. It runs on its own real-time operating system, and takes care of protocol conversions between the 3BNET and SNA/SDLC standards, relieving the 3Bs and IBM host of that task. Contained in the processor's control unit are its own processor, disk drive, memory, and special feature cards. There are two models, Model 1 emulating a channel or local connection to the IBM host; and Model 2, designed for remote applications, emulating a 3270-type cluster controller.

Note that computers running earlier releases of System V must be upgraded to swapping versions of Release 2.0 in order to support a communication processor. Note also that the product is being

introduced on a controlled basis only, and won't be generally available until the first quarter of 1986. And one final note: if budgets are your concern, concerned you well may be—the communications processor sells for \$27,000.

With the objective of enabling System V functions to be supported on IBM and compatible mainframes, AT&T has released System V-VM. The product is AT&T's version of Amdahl Corp.'s UTS/V, and is a complete System/370 implementation of System V.2. Both Amdahl and AT&T had a key part in developing the product, working for about a year on upgrading UTS for this specific purpose. The companies will continue working together to enhance the system. System V-VM provides full screen usage of 3270 terminals, up to 16 MB of user address space, virtual I/O capability, and other large system-oriented enhancements. General availability of System V-VM, in binary or source package, is set for October, but only on a year-by-year lease.

Jay Peterson, AT&T Supervisor for UNIX planning and product management, and Vish Vishwanath, UNIX Product Manager for System V-VM, were asked to compare V-VM with IBM's VM-UNIX product, VM/IX. V-VM, they said, is based on System V Release 2, and therefore offers a higher degree of compatibility with newer UNIX-related products. Second, V-VM provides support for full-duplex ASCII terminals and so can run interactive programs, something VM/IX cannot do. Also, V-VM contains Documenter's Workbench and other software, including some Berkeley features, not found as part of VM/IX.

David Buck, chairman of DL Buck and Associates, a company that supplies hardware manufac-

IBM PC AT XENIX TAPE

- * 45/60 MEGABYTES
- * 90 IPS STREAMING
- * HIGH SPEED SOFTWARE
- * NOW ONLY \$1095*



SHIPPING TODAY

The Bell XTC™ Xenix Streaming Tape System for the IBM PC AT and compatibles provides 45 or 60 Megabytes of high performance tape backup using ANSI standard 1/4" tape cartridges. 100% Xenix and IBM AT compatible. Production units are shipping today.

PEACE OF MIND THROUGH IBM QUALITY STANDARDS

The Wangtek 5000E unit at the heart of the Bell XTC system is the same high performance streaming tape drive used in the tape backup system sold by IBM. You get worldwide maintenance, economy of scale, and quality control driven by IBM standards, the highest in the industry. Don't settle for anything less.

HIGHEST PERFORMANCE HARDWARE

The XTC drive streams at 90 inches per second with data throughput up to 5 megabytes per minute. Bell's XTC optimizes performance with on-drive microprocessor, full DMA data transfer, automatic read-while-write ECC and smart controller data buffering.

UNIX SOFTWARE THAT MAKES DOS LOOK SLOW

Our staff has written over 50 tape device drivers on all of the Unix releases and has ported the entire Unix system many times. In addition to standard IBM Xenix tar, dump, backup and restore, we provide Bell proprietary high speed utilities for lightning fast backup. DOS software available too.

EASY TO INSTALL - INTERNAL OR EXTERNAL UNITS

Procure the Bell XTC in the internal IBM PC AT version for quick and clean installation, or as an external unit complete with AT-matching cabinet. We provide all cables, software and everything else needed for installation in minutes.

Bell Technologies

Post Office Box 8323, Fremont, California 94537
Sales Office: (415) 792-3646

*One time introductory unit price valid for prepaid orders received during September and November 1985. Limit one per customer, internal mount units only. Regular price \$1695, with volume purchase. OEM and reseller discounts available. Call Bell Technologies for best price and highest quality. Order your introductory unit today.

Circle No. 233 on Inquiry Card

turers with UNIX applications and drivers to interface their systems with IBM systems, offers another perspective on V-VM. AT&T has been under pressure for some time to put out a virtual memory product, Buck said. Others already have gone ahead and done it under UNIX, but many nevertheless are pleased to see AT&T now bless the effort. Comparing V-VM with IBM's VM/IX, however, is a bit like comparing apples (no pun intended) and oranges. The products have similar names but somewhat different purposes. "V-VM allows people to make good use of microprocessors", whereas "mainframe users (running VM/IX) are not precluded from doing UNIX".

The emphasis of AT&T's cluster of recent product announcements is on communications, the company's traditional strength; and, of course, the fact that it has now officially adopted links with IBM hardware is a welcome sight to Fortune 2000 customers. Only time and customers will tell, though, how significant these products will prove to be in the greater scheme of computer company jousting. The products may or may not determine if AT&T's computer business is still considered by some to be "fledgling".

WORK—BUT DON'T SMIRK—STATIONS

*"Someday, girl,
I don't know when,
we're gonna get to that place
we really want to go
and we'll walk in the Sun,
But till then, tramps like us,
baby, we were born to run".
Bruce Springsteen*

Doubtless Mr. Springsteen was not making an ambiguous reference to Sun Microsystems in composing these lyrics. He nonetheless conveys a sense of hope

and desperation, and these feelings may well be growing at various workstation manufacturer sites around the country. This summer has been witness to

**While enhancements
to the 3B series are
welcome, a point not
immediately clear is
which specific
customer markets
AT&T is targeting with
these upgrades.**

many technologies making the all-important business transition from rumors on research to machines coming off the line ready for sale. A primary example of this is the growing number of graphics workstations for engineering and scientific applications now appearing or scheduled to appear soon in the micro marketplace. The competition is heating up, and customers will have to start squinting to see past the smoke to find the product that will suit them.

Before deciding which workstation to buy, it's helpful to have a general notion of what composes a workstation. As impressive as the new 68020 microprocessor is, simply putting one in an old-model microcomputer does not a workstation make. UNIX REVIEW has looked at workstations before (January and July, 1985), and a working definition was offered in those issues. Richard Morin, one of our columnists and author of the January

article, "The Future of the UNIX Workstation", points out the basic components: a graphics terminal with high speed, high resolution screen and communications; an added processor; added memory; and added software. There are also certain traits or capacities based on the "four Ms": Mouse (or other pointing device), Megabyte (1 MB of RAM), Million Instructions Per Second (1 MIPS), and (1) Million pixels. Though helpful for listing workstation traits, current industry demands are such that the Ms should now be modified: for engineering and scientific applications, 2-4 MB of RAM are required; it's also beneficial to run at a speed somewhat faster than 1 MIPS. (A 68020 runs at 1.5 MIPS or so.)

Given this, consider the new offerings put forth by some substantial players. In the good old days (that is, a matter of months ago), there was Sun, Apollo, Cadmus, and Integrated Solutions. These four are still in the game (all are upgrading their products), but now there's also the DEC VAXstation II, the HP 9000 Series 300, the MassComp 500-Series upgrades, the Tektronix 6000 series and Tekstation AT, and even another Sun, the 2/130. Surely, there are more to come.

DEC's VAXstation II was assessed by Mark Sobell in his July column, so we proceed here to a description of the HP 9000 Series 300, currently available to OEMs and ISVs. Hewlett-Packard is taking the modular approach with the series, offering a choice of CPUs, displays, systems software, and peripherals. This can make comparison difficult, but the variety allows the customer to tailor the system and plan for future upgrades. Glenda McCall, HP Product Marketing Engineer, tells of four pricing bundles prepared for what HP considers two main markets—measure-

Making UNIX Easier with TEN/PLUS

Easier to Learn

Get your users started on UNIX by teaching them the standard keyboard commands and ten special commands. In the TEN/PLUS environment, that is enough to perform most common tasks and to invoke any application. As they gain experience, your users can employ more powerful TEN/PLUS commands and all UNIX commands.

Easier to Use

The procedure for using TEN/PLUS is simple: point at data with a cursor and then use a TEN/PLUS command. If your users need some prompting, they can ask for a menu. If they are confused, they can ask for a HELP message. If they are processing several files, they can open a window on each file.

Easier to Support

The TEN/PLUS environment eliminates the errors your users make when different applications require different sets of commands. Designed around a full-screen editor, TEN/PLUS allows users to manipulate all data with the text editing commands that they use most often. This means that users can use already familiar techniques to process both text and data. In addition, the TEN/PLUS system provides self-explanatory error messages.

Here's another way to reduce support costs: adopt TEN/PLUS as a standard user environment on a variety of computers—from personal computers to mainframes. Then provide TEN/PLUS to all kinds of users: clerks, managers, and engineers. A common user environment means your computer staff will have fewer products to support.

Easier to Network

We're porting the TEN/PLUS environment to most versions of UNIX and to VMS. We'll help you port it to other systems.

To link systems running our software, we're offering electronic messaging (INmail) and a network manager (INnet) as TEN/PLUS options. These packages are already a part of IX/370, the IBM mainframe UNIX system, and are available as an option on PC/IX, the IBM UNIX system for PCs. Thus, your TEN/PLUS system can readily participate in a network with IX/370 and PC/IX.

Easier to Expand

We're also offering a set of development tools that helps expand the TEN/PLUS environment. One kit provides your programmers with utilities and languages for defining and using screen forms, a user-friendly interface to the C compiler, and subroutine libraries. Another kit provides forms design and programming capabilities for your end users.

Call today for more information about the TEN/PLUS environment: (213) 453-UNIX.



Software Tools for System Builders.

INTERACTIVE
SYSTEMS CORPORATION

2401 Colorado Ave., 3rd Floor
Santa Monica, CA 90404
TWX 910-343-6255; Telex 18-2030
Telephone (213) 453-UNIX

ment automation (requiring only low-end packages) and design automation (where high-end specs are necessary). The least expensive bundle, selling for \$5750, comes with a 68010 CPU, 1 MB of on-board RAM, a bit-mapped display interface, four slots, and HP-IB, RS-232-C, and HP-HIL (human-interface link) interfaces. The design automation bundle, selling for \$27,725, comes with a 68020 CPU, a 68881 floating point processor, 2 MB of RAM, high resolution color graphics screen (1024 x 768 pixel board), eight additional slots, and the same interfaces as with the 68010. The customer is also allowed to spend as much as \$55,000 on the high-end of the

components scale. HP-UX can be run on all models.

It will be interesting to see how the Series 300 alters the composition of the HP 9000 family. Considered an upgrade of the Series 200, the 300 could become popular and make some of the 200 models (except the 216 and 236, which are desktops) obsolete. The Series 500 has multiprocessor capabilities, so one may consider software providing Network File Transfer service between the 300 and 500. Series 300 and 500 systems are also source-code compatible.

HP is making it clear that it intends to offer competition in the workstation market. Last March it announced an agreement with

Ceracor, Inc., a small, CAE software tools developer based in Salt Lake City, to integrate Ceracor's CDA 5000 software packages into HP's logic design products. Then, in July, HP acquired an 11 percent equity interest in Ceracor. According to William G. Parzybok, HP vice president and general manager of the Design Systems Group, "Ceracor's technology will be a strong element in HP's overall CAE product strategies."

Two companies, MassComp and Tektronix, have come out with package upgrades, MassComp for their own products and Tektronix for IBM. MassComp has two Performance Enhancement Packages (or PEPs—clever, no?) which serve as field upgrades for both the MC-500 and WorkStation-500 Series. PEP-501 consists of a 16 MHz, 68020-based CPU, 68881 floating point co-processor, a 32-bit Memory Interconnect Bus, and 2 MB of memory incorporating 256 KB memory chips; this package sells for \$7900. PEP-502 is PEP-501 plus a board-level floating point processor in place of the 68881, and sells for \$12,900. Though officially announced, the PEPs won't be available until "the first half of 1986".

The Tekstation AT, an IBM PC-AT with added hardware and software, is billed by Tektronix as a "low cost addition" to their 6000 Series of graphics workstations. Released through Tektronix's wholly-owned subsidiary, CAE Systems, Inc., the Tekstation AT incorporates a NS32016 co-processor, CAE 2000 design software, and like all machines in the 6000 Series, runs under UTek, a 4.2BSD-System V.2 hybrid version of UNIX. This AT concurrently supports PC-DOS and can perform the standard tasks of a PC, but under UTek can function through networking software as a work-

Great-looking TROFF output from low-cost laser printer!

■ TEXTWARE now offers DWB ■

For several years, Textware has been licensing TPLUS[†] software to process the output of TROFF and DITROFF for a wide variety of phototypesetters, laser printers, etc. Now we are pleased to announce the availability of Documenter's Workbench[‡] with all our packages. All TPLUS users may now benefit from this current TROFF offering, with even greater power and flexibility.

Many organizations are now getting maximum benefit from the HP LaserJet, using our TPLUS/LJ software. The low-cost LaserJet is a remarkable value on its own—8 page per minute output speed, 300 dot per inch resolution, and typesetter-quality fonts. TPLUS gives you access to all this *and more* from your own system. We support all the characters and accents needed by TROFF and EQN; in addition, special characters (☺; logos too) can be supplied or generated to meet specific requirements. Our precise handling of rules and boxes allows you to take full advantage of TBL for forms, charts, etc.

EQN examples	
$\lim_{z \rightarrow \pi/2} (\tan x)^{\sin 2x} = 1$	
$\frac{\alpha + \beta}{\sin(x)}$	$\prod_{k \geq 1} e^{S_k z^k / k}$

While even LaserJet output is not in the same class as the best phototype, it is certainly well suited to documentation and a broad range of other applications. When you do have a need for phototypeset images, TPLUS and the LaserJet will save you time and money. *Preview mode* lets you proof *all* aspects of your documents conveniently, in-house, before sending out for phototypesetting (from our UNI•TEXT service). Cross-device proofing is a standard feature of TPLUS.

The HP LaserJet printer is not only inexpensive—it is an exceptional value! Want proof? This entire ad was set in position using TPLUS on the LaserJet!

† TPLUS is a trademark of Textware Intl.

‡ Documenter's Workbench is a trademark of AT&T

For further information, please write or call.

Also available for:

- AM 5810/5900 & 6400, APS 5 & μ5, CG 8400 & 8600, Mergenthaler 202
 - Xerox 4045, 2700/3700 & 8700/9700
 - BBN, Sun, 5620 & 'PC' CRTs
 - Diablo, Qume & NEC LQPs
 - C Itoh & Epson dot-matrix
- Yes, TPLUS will support the new LJ.



TEXTWARE
INTERNATIONAL

PO Box 14 Harvard Square Telephone:
Cambridge, MA 02238 (617) UNI-TEXT

Circle No. 280 on Inquiry Card

"CrystalWriter... was the best and most logical choice for us."

Cari Laird Gray
System Administration
Health Science Center
University of Texas

"Thank goodness we placed the order! We are all very impressed with the product. We were thrilled when we discovered how easy it was, yet it had all the functions we needed and didn't insult our experienced word processor operators."

CrystalWriter word processing takes unsurpassed advantage of UNIX.[™] It's faster and it's easier to use, and it's per-

fectly at home in the multi-user environment. In fact, Syntactics' newest product, the *Crystal[™] Document Management System*, is designed to integrate UNIX word processing into office automation systems of tomorrow.

CrystalWriter is the next generation of UNIX word processing software.

"The first software to go through AT&T's certification testing in just one pass!"

AT&T Information Systems
Distributor and Publisher of CrystalWriter

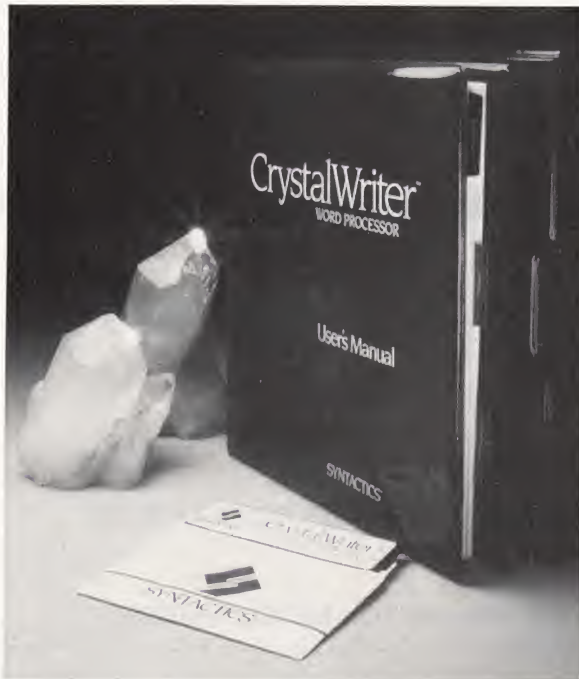
"A truly superior package."

UNIX/WORLD Magazine

When it comes to UNIX word processing software, make the same choice AT&T, UNIX/WORLD Magazine, and the University of Texas made.

For more information on CrystalWriter, call **(408) 727-6400** (within California); **(800) 626-6400** (outside California); or write: Syntactics Corporation
3333 Bowers Avenue, Suite 145
Santa Clara, CA 95054.

See us at UNIX EXPO, New York, Booth # 127



CrystalWriter[®]

The Word Processor of Choice for UNIX[™]



SYNTACTICS[®]

Circle No. 247 on Inquiry Card

CrystalWriter is now available on the AT&T UNIX PC 7300, 3B2, 3B5, 3B20; DEC VAX 750 & 780; NCR Tower; Plexus; Sun and many more. (The preceding are trademarked names.)

- UNIX is a trademark of AT & T Bell Laboratories.
- CrystalWriter and SYNTACTICS are registered trademarks of SYNTACTICS Corporation.
- Crystal is a trademark of SYNTACTICS Corporation.

station in team-engineering efforts. Such versatility comes with a price tag—Tekstation AT starts at \$25,000. CAE Systems is an IBM value-added dealer, and can add a 10 MHz co-processor, graphics card (for 720 x 704 pixels resolution), color display, and increase disk storage to 280 MB and RAM to 4.5 MB. (CAE also produces a 12-inch Liquid Crystal Shutter (LCS) color display for the AT; due out in October, it will sell for \$3700.) There is even a configuration complete with file server for about \$40,000.

THE SUN ALSO RISES

As a public relations statement will tell you, "Sun Microsystems designs, manufactures, and markets high-performance, general-

purpose workstations for technical professionals". The workstation is Sun's business, and Sun is making it its business to stay on top of the workstation market. For this purpose, this summer Sun has made notable changes in its marketing and product approach.

The first of these was across-the-board price cuts on its systems and memory. The 2 MB diskless Sun-2/50 now lists for \$8900, down from \$13,400; the Sun-2/160 Color SunStation with 2 MB of available memory was reduced from \$36,400 to \$27,900; and the 71 MB drive with 45 MB, 1/4-inch cartridge tape option available for the Sun-2/120, 130, and 160 pedestals dropped from \$10,900 to \$7900.

The price for 1 MB of memory, formerly \$4100, is now \$1500.

John Hime, director of product marketing at Sun, points out two main reasons for the price reductions. "Component costs are way down over the past year," for one. For another, though, as Hime admits, "There is a general heating up in the industry. Our competitors are continuing to offer discounts below their list price, and Sun needed to respond to this".

The second Sun move was to announce two new products: the Sun-2/130 SunStation, essentially a 2/160 without a color board and with monochrome monitor instead of color; as well as an optional mass storage subsystem for the Sun-2/50.

**UNIX
SYSTEMS
UTILITY
SOFTWARE
—SO YOU
CAN GET
ON WITH
YOUR JOB.**

For more information,
call or write.
(703) 734-9844

UBACKUP

BACKUP, RESTORE, AND MEDIA MANAGEMENT

USECURE

SYSTEM SECURITY MANAGEMENT

SPR

PRINT SPOOLING AND BATCH JOB SCHEDULING

SSL

FULL-SCREEN APPLICATION DEVELOPMENT

S-TELEX

TELEX COMMUNICATIONS MANAGEMENT

SSE

FULL-SCREEN TEXT EDITOR

.....
These products are available for most UNIX or UNIX-derivative operating systems, including System V, 4.2 BSD, 4.1 BSD, Xenix, Version 7, System III, Uniplus, and others.

.....
UNIX is a trademark of AT&T Bell Laboratories.

UNITECH

SOFTWARE INC.

Visit us at: **UNIXEXPO Booth No. 264 FCC Booth No. 675**

8330 OLD COURTHOUSE RD. SUITE 800 VIENNA, VIRGINIA 22180

Third, Sun announced upgrades to the 68020 processor for its VME bus-based products, the 2/130 and the 2/160. This announcement came in June, but the upgrades, available to new customers and past purchasers, will begin shipping at year's end.

There are two further points on the general development of workstations. The first concerns buses. With the emphasis on the 68020, a 32-bit processor, there is a tendency toward full 32-bit buses. The VME bus is gaining popularity, and for good reasons, according to Sun's John Hime: it's a full 32-bit bus, has a high bandwidth (20 MB/second transmission), and add-on board vendors are really going for it.

The other trend in workstation development regards the Motorola 68020 processor. It's been announced, it's been touted, and customers are looking forward to using the two to threefold increase in power it can deliver. Availability to customers, though, is something that takes time.

Motorola's Jeff Nutt, Technical Marketing Manager, and Jim Farrell, Manager of Technical Communications, state that their company was starting sample quantities in June of 1984, is at high production output at this writing, and is projecting production of 50-70,000 of the processors this year. Workstation manufacturers then have the ball in their court: retesting software (though programs running under the 68010 should run as is under the 68020), reprinting manuals, and retraining salespeople. Farrell also points out that many companies are taking the opportunity to enhance their operating systems at the same time they upgrade to the 68020, and such enhancement can push the product shipping date back a bit further. There are machines incorporating the processor that

have completed these procedures or soon will. The Altos 3068 started shipping in July, and HP was offering six to eight weeks delivery ARO for the 9000 Series 300 as of August 1. The 68020s to

be offered by many other manufacturers should be in their housings by Christmas.

David Chandler is the Associate Editor of UNIX REVIEW. ■

EASIER THAN 1-2-3...

BUT DESIGNED FOR LARGER SYSTEMS



P.O. BOX 2669
KIRKLAND, WA 98033-0712

EFFECTIVE SOFTWARE FOR BUSINESS



It's simple, C-CALC from DSD Corporation is more flexible, has more functions, and is easier to use than the best selling spreadsheet. We made it that way for a very simple reason, you'll get more work done and make better decisions in less time. That's what makes you successful whether you are planning for the future, forecasting trends, or analyzing profits.

The most popular spreadsheets require a great deal of time to get up and running. When we created C-CALC we kept in mind that time is your most important resource. Our On-Line Help facilities, prompts and menus allow even someone with minimal experience to see meaningful results in very little time. Our built-in training procedures let you pace your own learning with tutorial topics that range from basic to advanced. As you become more experienced, C-CALC allows you to bypass prompts and menus to save even more time.

So call DSD Corporation at (206) 822-2252. C-CALC is currently available for: UNIX, VMS, RSTS, RSX, IAS, P/OS, AOS, AOS/VS (Data General), IBM CSOS.

C-CALC is a registered trademark of DSD Corporation. UNIX is a registered trademark of Bell Labs. P/OS, RSTS and RSX are registered trademarks of Digital Equipment Corporation. AOS and AOS/VS are registered trademarks of Data General Corporation.

Circle No. 283 on Inquiry Card

THE HUMAN FACTOR

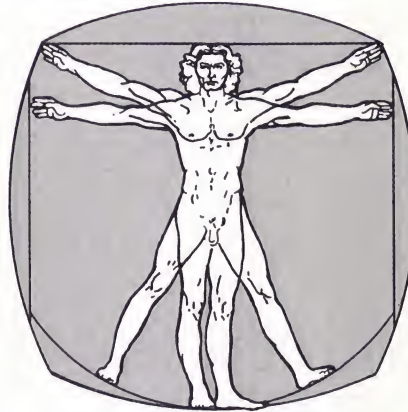
Alternative prototyping languages

by Richard Morin

When all you have is a hammer, everything begins to look like a nail.
Unknown

UNIX shells and utilities are very useful for fast prototyping. Previous columns (May and August, 1985) have touted them and given examples of their use. Part of their power lies in the fact that they are tuned to certain kinds of applications. The same tuning makes them less suitable for other tasks, but a well equipped workshop should maintain a variety of tools. This column presents a smorgasbord of prototyping languages, some of which may well belong in your toolbox.

Prototyping languages should promote interactive development, both by quick turnaround and by relatively loose structure. This means that, though they may have compilers, they should also have interpreters and/or incremental compilers. The very strong type checking and formal specifications required by many modern languages make them too demanding for fluid program development. Verbosity is also a bad thing in prototyping; a COBOL interpreter would not qualify. Finally, though specialization is often useful and even necessary, a prototyping system easily can become too finely tuned, turning into more of a parameterized



utility than a true programming language.

Nevertheless, there are a surprising number of languages meeting the preceding criteria. Even restricting the search to those that run on my Sun, I have found implementations of APL, BASIC, FORTH, Lisp, MAGIC/L, Nial, and Prolog. This list, while most assuredly incomplete, spans a wide range of programming language philosophies. Each language has its own peculiar way of looking at data structure, syntax, and other programming questions. This tends to suit certain languages to certain applications, or at least to certain kinds of users. The biases are not accidental, however, stemming directly from the way in which computer languages are developed.

Computer languages are invented mostly by individuals, sometimes by small groups, and occasionally by massive committees. The number of people involved in the process has a direct effect on the resulting product. Languages invented by individuals tend to be very elegant, pure, and cohesive, reflecting the intense effort expended in their creation. As larger groups enter the process, either in the design phase or as users, the language trades some of its purity for practical but inelegant additions. Massive committees seem to produce massive languages such as Ada and PL/I. Prototyping languages are no exception to this trend, as the following descriptions will show.

BASIC

BASIC (Beginner's All-purpose Symbolic Instruction Code) was invented in the mid-'60s by Drs. John Kemeny and Thomas Kurtz of Dartmouth College. Designed as an instructional tool, it has a simple algebraic syntax much like Fortran. This simplicity, and the ease of implementing BASIC interpreters, have made the language extraordinarily popular with amateur programmers, and almost ubiquitous on small computer systems. Unfortunately, this same simplicity has necessitated a plethora of changes

YOU CHOOSE:

	MLINK	CU / UUCP
Terminal Emulation Mode		
Menu-driven Interface	Yes	
Expert/brief Command Mode	Yes	Yes
Extensive Help Facility	Yes	
Directory-based Autodialing	Yes	
Automatic Logon	Yes	Yes
Programmable Function Keys	Yes	
Multiple Modem Support	Yes	Yes
File Transfer Mode		
Error Checking Protocol	Yes	Yes
Wildcard File Transfers	Yes	Yes
File Transfer Lists	Yes	Yes
XMODEM Protocol Support	Yes	
Compatible with Non-Unix Systems	Yes	
Command Language		
Conditional Instructions	Yes	
User Variables	Yes	
Labels	Yes	
Fast Interpreted Object Code	Yes	
Program Run	Yes	
Subroutines	Yes	
Arithmetic and String Instructions	Yes	
Debugger	Yes	
Miscellaneous		
Electronic Mail	Yes	Yes
Unattended Scheduling	Yes	Yes
Expandable Interface	Yes	
CP/M, MS/DOS Versions Available	Yes	

MLINK™

The choice is easy. Our MLINK Data Communications System is the most powerful and flexible telecommunications software you can buy for your Unix™ system. And it's easy to use. MLINK comes complete with all of the features listed above, a clear and comprehensive 275-page manual, and 21 applications scripts which show you how our unique script language satisfies the most demanding requirements.

Unix System V
Unix System III
Unix Version 7

BSD 4.2
Xenix
VM/CMS

MS-DOS
CP/M
and more...

Choose the best. Choose MLINK.

Altos
Arrete
AT&T
Compaq

Data General
DEC
Kaypro
Honeywell

IBM
Onyx
Plexus
and more...

MLINK is ideal for VARs and application builders. Please call or write for information.



Corporate Microsystems, Inc. P.O. Box 277, Etna, NH 03750 (603) 448-5193

MLINK is a trademark of Corporate Microsystems, Inc. Unix is a trademark of AT&T Bell Laboratories. IBM is a registered trademark of IBM Corp. MS-DOS and Xenix are trademarks of Microsoft Corp. CP/M is a registered trademark of Digital Research.

Circle No. 281 on Inquiry Card

to BASIC, with virtually no standardization.

It has also made BASIC an object of scorn for professional programmers, who ridicule its limited control and data structures, lack of modularity, and unimpressive performance. Fortunately, many of these problems are being addressed. Compiled BASIC can be quite efficient; extensions allow for modern language features, and standardization efforts are underway. In the meanwhile, BASIC devotees at least can gloat about the enormous quantities of public domain software available to them.

APL AND NIAL

APL and Nial ("Nee-al") are characterized by the use of arrays as their fundamental data structure. While other languages *have* arrays, these languages *use* them, allowing them to be manipulated as easily as scalars. Both languages support nested, heterogeneous arrays, allowing elements to be of any type, including arrays. APL and Nial have operators that are roughly equivalent to the built-in functions found in C and Fortran. Since the operators work on arrays as well as scalars, however, their power is much greater.

APL (A Programming Language) was invented by Dr. Kenneth E. Iverson of IBM in the early '60s. Often characterized as a "write-only" language, APL makes C look verbose. The most famous features of the language, in fact, are its terseness and the plethora of special characters it uses. These are closely related—the special characters actually serve as operators in APL. The APL character set is thus a mixture of Roman, Greek, and mathematical symbols mapped onto a standard keyboard. The growth of APL has been severely hampered by the need for special display

and printing devices. With the advent of bitmapped displays and intelligent printers, APL may get a second chance for glory.

Nial (Nested Interactive Array Language) was developed in the

The very strong type checking and formal specifications required by many modern languages make them too demanding for fluid program development.

late '70s by Dr. Michael A. Jenkins of Queen's University and Dr. Trenchard More, Jr. of IBM. Based on array theory, as developed by Dr. More in the late '60s, it has been implemented as a commercial product named Q'Nial. Nial uses operators much like APL, but they are written as alphanumeric tokens. In addition, Nial is able to embed its operators in data structures, offering Lisp-like capabilities.

Nial advocates argue that it is a more complete language than APL, and is therefore more powerful and consistent. This consistency, they say, allows Nial programs to be transformed and analyzed formally in ways that are impossible for other languages. The syntax, in any case, is a hybrid, looking a bit like a mixture of APL and C. This is due to the fact that Nial operators can be "applied" to values, "composed" with other operators, and

generalized by built-in "transformers". This allows Nial operators to function on the almost limitless variations of data structures allowed by the nested heterogeneous array format.

Don't expect blinding efficiency from either APL or Nial, since they are typically (though not always) implemented as interpreters. Instead, think of them when your problem has a strong mathematical or statistical flavor, and consider using Nial for AI applications. Most programming languages treat arrays as places to store things, and only allow work to be performed on the things themselves. If this part of Von Neumann's legacy is cramping your style, consider trying one of these more mathematically oriented languages.

FORTH AND MAGIC/L

FORTH and MAGIC/L ("magical") are similar in several respects. The principal difference is that MAGIC/L has dropped FORTH's RPN (Reverse Polish Notation) syntax in favor of a more conventional algebraic style. Both languages are implemented as incremental compilers, giving them nearly the speed of conventional compiled code, and most of the convenience and flexibility of an interpreter. While both languages run well under UNIX, they can be used without it, and often are. Finally, the languages show similar facility for dealing interactively with hardware devices.

FORTH was developed by Dr. Charles Moore in the early '70s. Its principal data abstraction is the stack, but it supports almost anything else on a roll-your-own basis. The RPN syntax is the principal target of FORTH critics. They note that it requires one to remain aware of the state of the stack at all times, and that it makes reading FORTH code quite

difficult. FORTH advocates argue that awareness of the stack encourages efficient use of the machine, and that the simplicity of the syntax allows it to be very general and powerful. In any case, FORTH is flexible enough to allow any determined user the chance to modify its appearance.

MAGIC/L, invented by Arnold Epstein and Jeffrey D. Morris of Loki Engineering, is a well developed alternative solution, however. It is incrementally compiled, handles machine dependencies beautifully, and looks surprisingly normal, rather like simplified C or Pascal. Although MAGIC/L only supports singly-subscripted arrays, its "record" construct allows the definition of fairly arbitrary data structures.

LISP AND PROLOG

Lisp (LISt Processing) and Prolog differ radically from each other, but they are both heavily used by the AI community. Their strongest similarity is perhaps their facility for symbol manipulation. This facility suits them for working with knowledge (facts, rules, relationships, and the like) as opposed to mere data. Since AI coding is done in a highly interactive and exploratory manner, both languages have a strong prototyping flavor. Additionally, both languages use lists as their basic data structure, though the details differ widely.

Lisp was invented by Dr. John McCarthy and his associates in the late '50s, but it has undergone drastic changes since that time. Although Lisp was strictly an interpretive language initially, implementations now generally compile or interpret as desired. Both programs and data are stored by Lisp as binary trees, and recursion is the main vehicle for traversing the trees. This simple and consistent structure has proven to be remarkably

flexible, allowing the grafting of a wide range of programming paradigms onto Lisp. Lisp programs can, for instance, contain "intelligent" data structures or generate expressions that are then evaluated. AI programmers commonly create problem-oriented dialects to assist them in their work, much as UNIX devotees create tools such as **sed** and **make**.

Prolog was developed in the '70s by assorted computer scientists in Marseille (Battani, et al.), with help from Warsaw (Kluzniak, et al.), Edinburgh (Pereira, et al.), and elsewhere. It differs from conventional programming languages in that its semantics are declarative rather than imperative. One thus programs Prolog by giving it a set of facts and rules and asking it to prove a given assertion. Prolog scans its knowledge base, attempting to find a combination of facts and rules that will provide the requested proof. Part of the interest in Prolog stems from the possibility that hardware could be built to perform these searches in parallel, providing an opportunity for efficient large scale multiprocessors.

IN SUMMARY

For every class of programming application, there is likely to be someone who will get fed up with the existing languages and build another. Alternatively, a language may be developed simply to experiment with ideas or to express a philosophical point. Most of these fade into (often well deserved) obscurity, being too specific, insufficiently robust, or simply too similar to other available tools. Occasionally, however, a group of followers will develop around a language. Software is produced, books and articles are written, and a subculture develops. Some of the languages de-

scribed above have withstood this kind of scrutiny; others are still under the gun. All of them, however, have attracted users and advocates, and hence deserve consideration by adventurous programmers.

Mail for Mr. Morin can be sent to Santa Forda Computer Lab, PO Box 1488, Pacifica, CA 94044.

Richard Morin is an independent computer consultant specializing in the design, development, and documentation of software for engineering, scientific, and operating systems applications. He operates Santa Forda Computer Lab in Pacifica, CA. ■

FRANZ

THE FIRST NAME IN
LISP

Franz LISP from Franz Inc. is currently available under UNIX and VMS. Now with Flavors and Common LISP compatibility. Franz sets the standard for LISP.

Franz Inc.
1141 Harbor Bay Parkway
Alameda, California 94501
(415) 769-5656

UNIX is a trademark of Bell Labs. VMS is a trademark of Digital Equipment Corporation

Circle No. 278 on Inquiry Card

THE RIGHT LANGUAGE FOR THE JOB

An overview of the potpourri of UNIX options

by Joel McCormack

UNIX and C are intertwined symbionts that feed off one another's success. The fact that UNIX is written in C has made the system portable and extensible, which has helped make it popular; the C software base generated by UNIX in turn has spawned C compilers for other operating systems.

Thus, a discussion of programming languages available on UNIX must revolve around C. Why is C so dominant, and how have other languages filled niches not addressed by C?

This article describes how C allowed UNIX to be written in the first place, and explains why C remains the UNIX programming language of choice. The evaluations of Fortran, Pascal, Ada, and Lisp that follow emphasize the advantages these languages have over C, and the weaknesses that prevent them from displacing C.

The shell, which supplies the glue for sticking programs together, and **make**, which assists with the generation and maintenance of systems of programs, are then briefly described.

Finally, this paper delineates

C's deficiencies, and proposes a possible replacement: Modula-2. Offering the power and efficiency of C with none of the flaws, Modula-2 may ultimately emerge as the language to surpass C as a tool for writing and maintaining software systems.

C

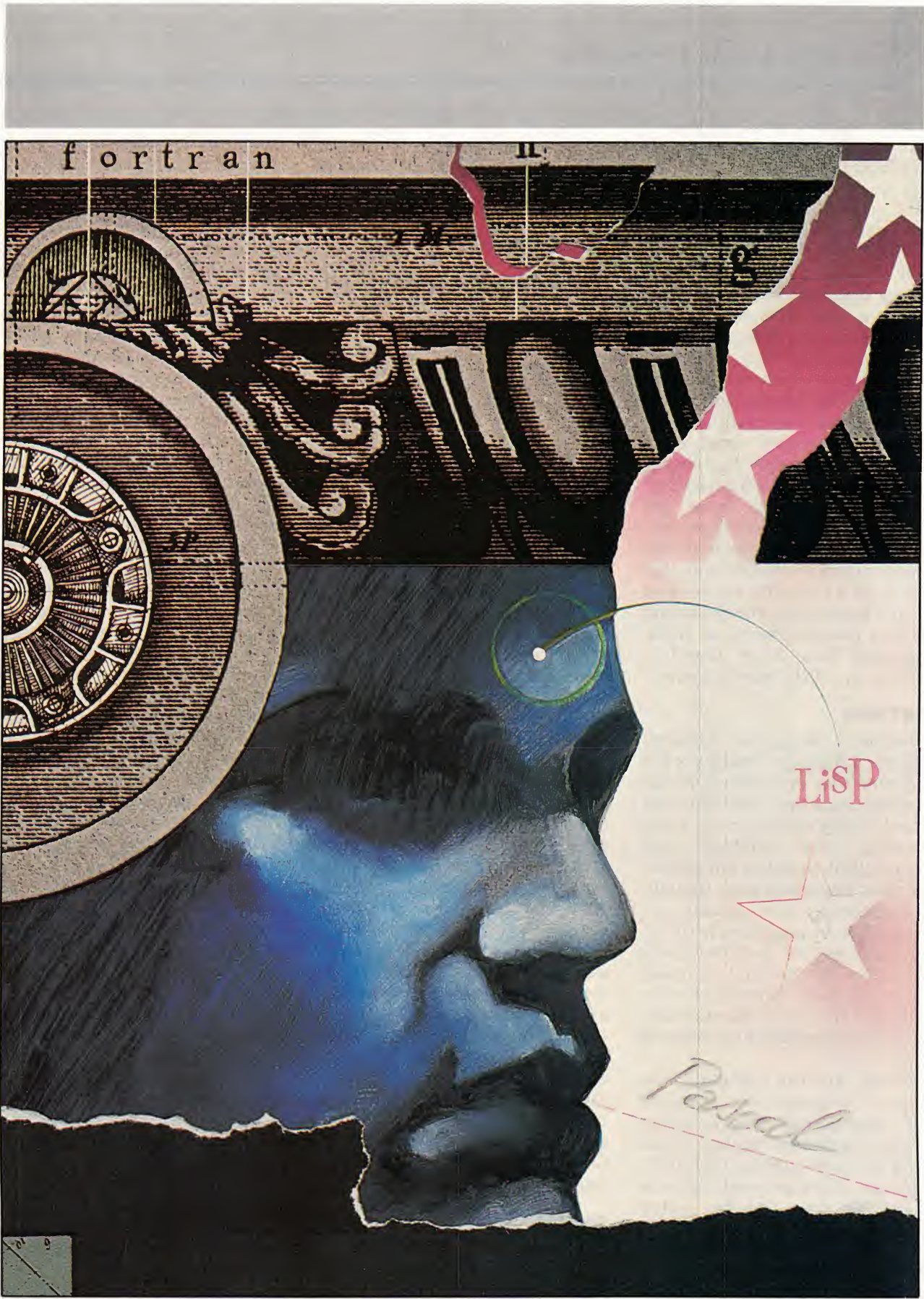
Though the original UNIX operating system was written in assembly language, its authors realized that further progress required a more powerful notation. Their experiences with the languages BCPL and B led Dennis Ritchie to include multiple primitive types and a method of composing types in the successor language he called "C".

C was intended only as a more convenient notation than assembly; efficiency and access to low-level facilities had priority over all else. Hence C does not allow the assignment of one array to another, and types are not included for compile-time checking, but merely allow the code generator to emit the proper instructions.

The language succeeds as a step up from assembly program-

ming. C can do nearly everything that assembly can, at a modest cost in efficiency. The Berkeley 4.2 release contains less than 1300 lines of assembly code in the kernel, while I estimate code written in C is no more than 40-80 percent larger than hand-written assembly code. C also supports the creation of libraries of routines. A program can be broken into pieces that are developed and compiled independently; general-purpose routines need not be incorporated textually into a program, but only linked to it.

C has two unique advantages over all other languages in the UNIX environment: it was the first language implemented, and remains the only system programming language available on all versions of UNIX. As a result, programmers often modify existing C programs rather than start from scratch in a language better suited to the task. And the benefits of a large software base now extend beyond UNIX: systems software companies, particularly in the microprocessor world, offer a variety of C compilers with "UNIX-compatible" libraries.





Considering the importance of C to UNIX, the portable C compiler is surprisingly mediocre. Its speed is not exceptional (1400 lines per minute on a VAX 11/780). The compiler tends to be easily confused by errors, often making it easier to fix one mistake at a time in preference to wading through a deluge of spurious complaints. The code the compiler generates also leaves much room for improvement.

To compete successfully with C, other languages must offer something C does not. This strong suit may be anything from a large engineering software base (as with Fortran), to language features not found in C (as with the strong typing of Pascal and Modula-2), to a view of a world alien to C (as with Lisp). The following sections describe some of these languages that have found a home in the UNIX environment.

FORTRAN

Fortran's designers intended that the language would permit algorithms to be written in an algebraic manner, and that the housekeeping operations of array indexing, flow control, and input/output would be simplified. The most important goal, though, was efficiency: John Backus expected Fortran programs to run at least half as fast as hand-coded assembly. The original compiler performed such impressive optimizations that it set the standard for Fortran compilers for years to come.

Today, Fortran remains the primary language for scientific and engineering programming. Fortran benefited greatly from being the first popular high-level language: there are vast libraries of Fortran that can be ported between machines as easily as C programs can be ported to other operating systems. And even though Fortran's only type struc-

Modula-2 may ultimately emerge as the language to surpass C as a tool for writing and maintaining software systems.

ture is the array and its control structures are anything but modern, it offers better support for arrays and real number types than C. Arrays can have variable bounds, and support for both single and double-precision real and complex numbers is offered.

Fortran is well integrated into Berkeley UNIX. It can call C routines (and vice-versa), the Fortran 77 standard is implemented, and two pre-processors, RATFOR and EFL, convert extended versions of Fortran into standard Fortran. But the Fortran compiler is less error-free than the C compiler, and the code it generates is unimpressive; two of the four persons who reviewed this article related stories of programming groups using VMS rather than UNIX just to gain access to the VMS Fortran compiler. Better Fortran compilers may appear as UNIX is implemented on the new generation of high-speed number-crunching machines, but we must wait to see.

PASCAL

Niklaus Wirth created Pascal in order to teach the clear expression of data composition and

structured control flow. The type system of Pascal is based on the ideas of another professor, C.A.R. Hoare, who in turn found Pascal coherent enough to produce an axiomatic definition. The language has since spread well beyond its academic roots.

Types in Pascal correspond to familiar mathematic abstractions, and are designed to help prevent common programming errors. Thus, pointers in Pascal refer to a specific type; the values that are assigned to scalars can be limited using subrange declarations; set operations are included; and arrays have declared lower and upper bounds. Most importantly, different types cannot be mixed freely.

Pascal is a far safer language than C to teach to beginning students. In C, something as mundane as a copy of one array to another may destroy all of memory if the loop terminating condition is written incorrectly. In Pascal, the same action is written either as a single assignment statement, or as a loop in which indices into the source and destination arrays are checked to ensure that they are within the bounds of the arrays.

Many seasoned programmers also appreciate the safety afforded by the strict type-checking of Pascal. Compared with C, Pascal programs that compile are more likely to be bug-free; if something goes wrong at runtime, a Pascal program is more likely to complain nicely than to behave in a mysterious way. Range-checking, case-checking, variant-checking, and nil pointer-checking enable Pascal to report an error in the vicinity of its source, rather than in a spot of the program many instructions later.

Designed for use by student programmers, the Berkeley Pascal compiler handles compile-time errors better than any other

compiler on UNIX. By using an extended version of the **yacc** parser generator, the Pascal compiler gives extremely accurate messages, and rarely gets so confused that it cannot continue parsing. It also emits a variety of warnings, detecting when a variable is used but never assigned, or when a variable is declared but never used.

Berkeley Pascal's extension for separate compilation offers a simple but effective version-checking scheme: the compiler computes a hash value for each *.h* file that's included and stores this value in the resulting object file; when the compiler links several object files into an executable program, it ensures that all object files have the same hash value recorded for a given *.h* file. Berkeley Pascal offers a few other extensions that alleviate Pascal's problems by blank-filling constant strings to the right size, and by providing I/O procedures for file opening, closing, and random-access. Like Fortran, Pascal can call C and vice-versa.

For all that, Pascal is still a weak contender against C. The Berkeley compiler compiles at 950 lines-per-minute, and under 800 lpm if runtime checks are generated. What's more, it generates poor code. And finally, whereas "standard" C is generally useful, standard Pascal often is not. Pascal's type-checking simply does not outweigh these disadvantages.

ADA

The Department of Defense, realizing that languages in use by the military were inadequate for reliably producing large software projects, commissioned four competing groups to design a language. This language was to support many advanced programming concepts, including type-checking, modularity, and

high-level support for low-level tasks. Eventually, the Defense Department decided on one of the proposals; after many revisions it became the language Ada.

The result is a powerful but frustrating language. Ada has two major problems: a "kitchen sink" feeling that stems from committee decisions about language constructs; and features that interact poorly and are hard to compile. Because implementations of Ada were not developed concurrently with the language specification, there was little practical feedback about design decisions.

As a result, Ada is not popular. It takes a while to learn to use it well, and compilers for Ada are large and slow. Ada on UNIX is used largely for education and to cross-compile programs for military computers.

LISP

Lisp has a view of the world quite unlike C's. Originally developed in the late 1950s and early '60s, Lisp was designed for the manipulation of symbolic expressions rather than numbers. Whereas Fortran is oriented toward a concrete realization of functions, Lisp is one step removed—it is often used to perform operations (like differentiation) on functions themselves. All operations, including arithmetic, are written in a functional notation based on Alonzo Church's lambda calculus.

A fluke in Lisp's design eliminates the boundary between data and code, allowing the language to execute the very lists it creates. Lisp is historically significant for another reason: it pioneered automatic garbage collection. These qualities have made the language the mainstay of the artificial intelligence community.

Lisp's longevity means that thousands of programs have

SVS FORTRAN

The SVS FORTRAN-77 language is now available on an expanded family of CPUs. It fully supports:

- Full Ansi Standard

MC68000

- GSA Certifiable
- Many Language Extensions derived from our large user community
- Symbolic Debugger

NS32000

- Optimizing Code Generation
- High Speed Compilation
- Very Large Applications
- Integrated Hardware Floating Point Interfaces:
MC68881, SKY, NS32081,
and Custom

MC68020

- Complete User Documentation
- Available since 1981

SVS has been a major OEM supplier of compilers since 1979.

For further information about these and other quality OEM compilers call 415/549-0535.



Silicon Valley Software, Inc

10011 N. Foothill Blvd., Suite 111
Cupertino, CA 95014

Circle No. 272 on Inquiry Card

TM

EPIX

SUPERMICRO FROM THE WORLD OF DEC

TOOL OF THE TRADE



Just as one good thing leads to another, AT&T's innovation of UNIX* led straight to DEC's enhancement of it, ULTRIX.**

Which now has led to the latest related advance. To the tool that makes ULTRIX software more useful than ever.

EPIX. The microcomputer that's so handy it's already becoming the tool of the trade.

Like any good tool, EPIX makes jobs easier. Because it not only comes packed with all the ULTRIX software, but supports everything else in the world that's QBUS compatible. Meaning there's a raft of helpful hardware readily available too.

And since programmers need to count on their tools, we've built EPIX with the most reliable industry-standard innards. From its UNIX time-sharing system to its J-11 super microprocessor.

To make EPIX exceptionally useful, we've also made sure users won't outgrow it. By giving it up to 369Mb of formatted high-speed Winchester capacity. And a main memory of up to 4Mb.

So you can make it fit the job now, and it'll still be able to grow as the work expands.

What's more, EPIX comes from a company that tries to be every bit as helpful as its products. A company more interested in what you want than what it's got.

To test that statement, you can simply use another tool of the trade. The telephone.

Call:
1-800-UNBOUND Toll Free
In California 714-895-6205
TWX: 510 100 1075

EPIX UNBOUND

Unbound Inc.
15239 Springdale St.
Huntington Beach, CA 92649

Circle No. 254 on Inquiry Card

*Trademark of Bell Laboratories

**Trademark of Digital Equipment Corp.

been written in it. Many of these are included in Berkeley UNIX's Franz Lisp, along with some options and support packages that make the porting of other Lisp software from various dialects feasible. Franz Lisp programs can be either interpreted or compiled, and can call C, Pascal, and Fortran routines. Franz Lisp also includes a nice debugger and a user-extensible editor.

On the downside, Franz Lisp is oriented toward teletype interactions, and executes slowly. Lisp on UNIX is nice to have if you require UNIX for other things, but doesn't quite satisfy the serious Lisp developer. Lisp may find more acceptance on UNIX as UNIX is ported to new, high-speed machines.

THE SHELL

Part of the UNIX philosophy is that programs should be small and single-minded; programmers can then hook them together (via pipes) to accomplish more complex tasks. The various shells provide a simple, flexible mechanism for wiring programs together. Two popular versions are the Bourne shell (**sh**) and the C shell (**cs**h).

Both **sh** and **cs**h are interactive. A single command line can pass string arguments to programs, redirect input or output to a file, pipe several programs together with the output of one going to the input of the next, and expand wild-card filenames.

Both shells are also used for programming more complex patterns of program invocation. **If** statements and **loop** statements test to see if a program or sequence of programs have terminated normally, and the **case** statement tests string arguments and variables using the same pattern-matching conventions as filename expansion does. The output of a program can even be

used as a statement of the shell program.

The use of shell programs (or scripts) fall into two general classes: one is to customize existing commands. Such scripts are but one or two lines, and add only a fixed set of arguments to those already provided, or pipe several programs together into one superprogram.

The other use of scripts resembles conventional programs. Such scripts use variables and control constructs, and range from half a page to several pages. These programs typically interpret the arguments passed to them, invoke different programs based on the arguments, and/or perform some operation on each file in a directory.

The use of scripts is transparent to the user. A UNIX command is an executable object file generated from C, Pascal, Fortran, Modula-2, or Lisp; or a script written in any of the shells. All are invoked and passed arguments in the same way. This uniformity and the easy access to pipes provides a meta-programming environment superior to nearly all other operating systems.

THE MAKE PROGRAM

Because of **make**, it is possible to maintain large programs on UNIX (including UNIX itself). The **make** program uses two kinds of rules. The first is a generic set of rules for deriving one type of file from another: for example, **.o** object files are created from **.c** source files by calling the C compiler **cc**. The second is a specific set of rules stating dependencies between a group of files; for example, an object file depends on the main **.c** source file that is compiled to create it, as well as all **.h** header files that are included. A specific rule may include a derivation command

SVS Pascal

The SVS Pascal language is now available on an expanded family of CPUs. It fully supports:

- Ansi Standard

MC68000

- IEEE Floating Point, both Single and Double Precision
- Full Featured with most UCSD Extensions
- Interlinkable with SVS "C" and SVS FORTRAN
- Symbolic Debugger

NS32000

- Optimizing Code Generation
- High Speed Compilation
- System Programming and Very Large Applications
- Modular Programming
- Secure Separate Compilation

MC68020

- Complete User Documentation
- Available since 1980

SVS has been a major OEM supplier of compilers since 1979.

For further information about these and other quality OEM compilers call 415/549-0535.



Silicon Valley Software, Inc

10011 N. Foothill Blvd., Suite 111
Cupertino, CA 95014

Circle No. 272 on Inquiry Card



that overrides the generic rule.

When told to ensure that a file is up-to-date, **make** looks at the dependency rule for the file. If the specified file was last created or modified more recently than all the files it depends on, **make** tells you all is well. Otherwise, **make** regenerates the file (after regenerating any files the specified file depends on).

The **make** program is merely convenient when used to specify the same set of compiler flags across all compilations; it is essential when the files depend on each other, since a change to one file may require recompilation of several others. The **make** program thus provides version-checking and automatic recompilation to languages that lack this capability.

THE PROBLEMS WITH C

UNIX is no longer a small operating system with a few text-processing utilities. I suggest that the very language that made UNIX possible is now slowing its further progress. Not all problems can be reduced to small programs connected via pipes, and C is an inferior programming language for large, multi-author software systems.

First, the notation itself is not error-resistant. Frequent C programming mistakes involve one-character mishaps, like substituting = for ==, & for &&, | for ||, or putting a semicolon after a **for** loop header. C programmers waste time tracking down simple typographical mistakes that other languages detect at compile time.

Second, the type composition rules of C are contorted for all but the most simple structures: declarations use both prefix and postfix notation with an arbitrary precedence hierarchy. The C declaration for a variable that is a function for returning a pointer to

I suggest that the very language that made UNIX possible is now slowing its further progress.

an array of pointers to characters is:

```
char    *(*(**f())[MAXSTRINGS]);
```

Third, the lack of type-checking is, I hazard, the largest source of productivity loss for C programmers. The portable C compiler generates warnings for many of the legal—but usually wrong—constructs that C allows, thereby encouraging the use of type casts, but many mistakes go unnoticed. The use of the type **int** to represent integer, character, and boolean types often leads to programs that manage to compile, but compute the wrong result.

The deficiency of type-checking extends to procedures as well. Parameter types are not part of the procedure header declaration but are only included in the actual procedure definition, so *.h* files do not contain type information. Worse yet, when a procedure accepts a variable number of parameters, the type of parameters that are expected can be determined only by reading the code inside the procedure.

Further, C's parameter-passing convention invites disaster. In languages like Pascal and Modula-2, the method of passing a parameter—by value or by address—is specified once in the procedure heading, and the compiler ensures that the correct code

is generated thereafter. C forces the programmer to remember and specify the appropriate method at each call to the procedure.


When these points are raised to staunch C defenders, their reply is "Well, you can always run **lint** to do all that do-gooder type-checking stuff." This answer invariably comes from programmers who don't use **lint** themselves; those who do know it is a weak substitute for type-checking at best, and an irritation at worst.

On one hand, many common errors slip by **lint** with nary a peep...after all, it may be wrong, but it is valid C. For example, **lint** does not complain about either of the following statements (*i* and *junk* are **int**):

```
if (i = 0) { ... }
junk = scanf("%d", i);
```

On the other hand, **lint** often acts like the boy who cried "Wolf!" For example, since **lint** can only check to see that you are using a procedure consistently, it complains at every call to a procedure that takes a variable number of parameters. You can tell **lint** not to check certain constructs, but this defeats the very purpose of the program. Even so, the bug list for **lint** in the manual is a single sentence: "There are some things you just can't get **lint** to shut up about."

Fourth, with no subranges and no type-checking, C is left with no way to perform runtime checking. Range-checking in particular is lacking; it is all too easy to index past the end of an array and destroy the variable(s) allocated immediately following the array. The behavior of a program subsequent to such an event often leads the hapless programmer down many false paths before the source of an error is finally located.



The lack of type-checking is, I hazard, the largest source of productivity loss for C programmers.

Fifth, C is not as portable as its proponents often maintain, simply because it has not been precisely defined. In their book *C: A Reference Manual*, Samuel Harbison and Guy Steele, Jr. write: "The second source of information on C is the C compilers themselves; you can write a C program and see if it compiles (and, if it does, what code is generated)." This diversity extends even to UNIX: the C compilers on 4.2BSD and AT&T System V accept different languages.

Finally, C has no support for modules in the modern sense of the word, **include** and **make** notwithstanding. True modularity provides version-checking, a non-global name space for exported objects, true separation of specification from implementation, and module initialization.

MODULA-2

Just as C eliminated many of the errors endemic to assembly programming, new languages eliminate many of the trivial (but costly) errors endemic to C. One such language, Modula-2, combines a type-secure module structure with the low-level program-

ming capabilities of C and the type-checking facilities of Pascal.

Modula-2 is the third in Niklaus Wirth's line of strongly typed languages. The first is Pascal,

which succeeded far beyond its design goals. The second is Modula-1, a small, experimental language designed to investigate modularity and concurrent programming. Modula-2 is a systems programming language synthesized from Wirth's experiences with both of his earlier languages and Mesa, a language developed at Xerox PARC.

The most important structure in Modula-2 is, appropriately enough, the module. Modules have two separately compiled parts: a *definition* module, which contains declarations of constants, types, variables, and procedure headings accessible to client modules; and an *implementation* module, which contains private declarations, code for the procedures declared in the definition module, and an initialization section.

In C, a change to a *.h* file may require that all clients be recompiled—but the responsibility falls to the programmer, who is armed only with **make**; in Modula-2, strict version-checking forces consistency. This checking can be implemented with simple time-stamps, which require recompilation of client modules any time a definition module is changed, or the checking can be implemented with more complex schemes that require recompilation only when non-upward compatible changes are made. Version control in Modula-2 does not depend on the careful (but possibly erroneous) construction and maintenance of dependency lists in a **make** file, but is automatic and failsafe.

In C, if two different *.h* files declare the same identifier, a client program cannot include

both files; in Modula-2, a client may import names from a module in either *qualified* or *unqualified* mode. References to qualified identifiers are prefixed by the

SVS BASIC-PLUS

The SVS BASIC-PLUS language is now available on an expanded family of CPUs. It fully supports:

- DEC BASIC-PLUS Dialect

MC68000

- Integer, String and IEEE Double Precision Variables
 - Vector and Matrix Arithmetic
 - String Arithmetic
- Extended I/O, including Print Using, Get and Put
 - Multi-lined Functions
 - Renumber, Trace, and Chain Commands

NS32000

- Very Fast Interpreter and Source Code Protection using quasi-compiled internal form
 - Interactive

MC68020

- Complete User Documentation
- Available since 1982


SVS has been a major OEM supplier of compilers since 1979.

For further information about these and other quality OEM compilers call 415/549-0535.

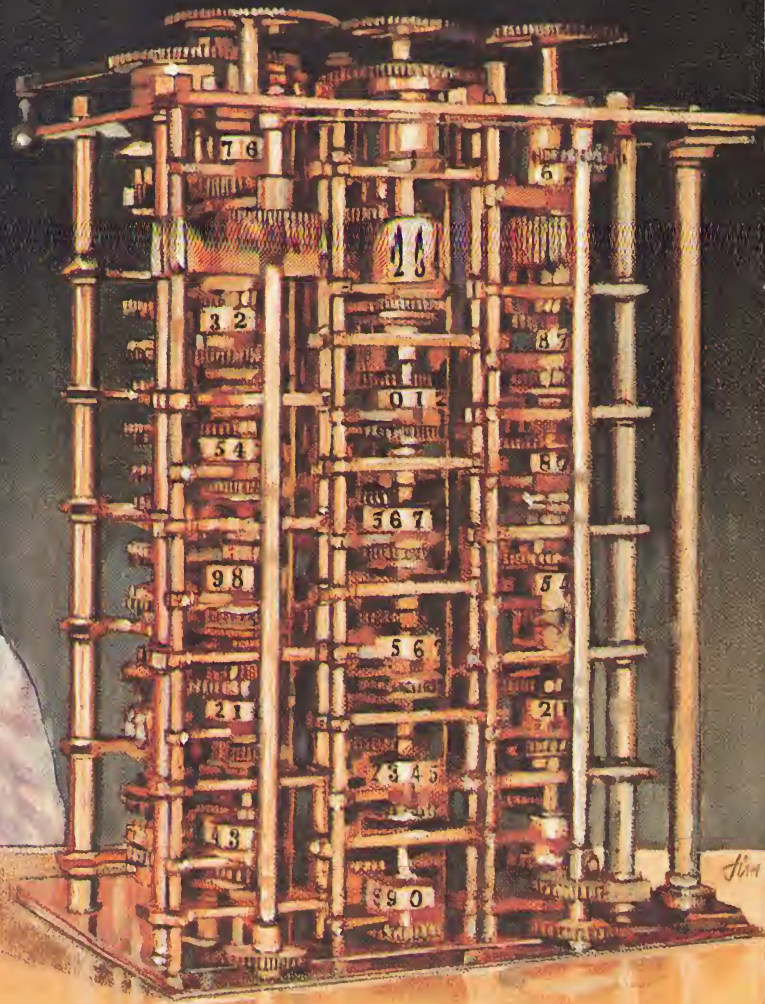


Silicon Valley Software, Inc

10011 N. Foothill Blvd., Suite 111
Cupertino, CA 95014



Circle No. 272 on Inquiry Card



Sorry, Countess, this is one computer where you won't find VADS™!

Although the VERDIX Ada® Development System (VADS) won't be rehosted on Charles Babbage's Difference Engine, it is being hosted on and targeted for a variety of computer systems and embedded system architectures.

The Department of Defense (DoD) has now validated VADS for a growing number of computers and operating systems including the DEC/VAX™ series under UNIX™ 4.2 BSD and ULTRIX™, and for the Sun-2™ Workstation. Future product releases will include Host Development Systems for VAX/VMS™ and UNIX System V, and cross-targeted systems for 4 major architectures... Motorola 68000 and Intel "86" families, the NS32032, and MIL-STD-1750A.

VADS is the fastest and friendliest Ada development

system available. It is specifically designed for large-scale Ada program development in a production environment.

VADS features a complete run-time system, plus an interactive, screen-oriented, fully symbolic debugger that lets you easily pinpoint errors. Unexcelled diagnostics and Ada library utilities quickly manage, manipulate and display program library information, dramatically shortening development times.

VADS from VERDIX. The finest, fastest and most cost-effective Ada Development System on the market today. The biggest breakthrough in programming since Ada herself.

For full information, call James Zimpfer, Director of Sales and Marketing Support, Ada Products Division, at (703) 378-7600.

VERDIX•

14130 Sullyfield Circle, Chantilly, VA 22021

Ada is a registered trademark of the U.S. Government, Ada Joint Program Office.
VAX, VMS and ULTRIX are trademarks of the Digital Equipment Corporation

UNIX is a trademark of Bell Laboratories
Sun-2 is a trademark of Sun Microsystems, Inc.

VERDIX and VADS are trademarks of Verdix Corporation

Circle No. 255 on Inquiry Card

name of the module in which they are declared (for example, *ModuleName.objectName*), freeing the module designer from futile attempts to provide objects with unique names.

In C, changing the implementation of one of the "modules" used by a main program may require changing and recompiling the program; in Modula-2, changing an implementation module requires recompiling only that module. Further, Modula-2 definition modules change less frequently than C .h files, due to Modula-2's support for abstract data types. The structure of an abstract data type is not known outside of the module, and the only operations allowable on the type are those provided by the module. Unlike C, changing the structure of such a type does not require recompiling modules that use the type.

In C, .h files that are otherwise irrelevant to a client program must be included by the client so that it can call the initialization procedure for each "module"; in Modula-2, initialization sections are called automatically by the client, which ensures that lower-level modules are initialized before the modules that depend on them.

Modula-2 has all the advantages of strict typing, but it adds the flexibility of C's type casting and pointer arithmetic. Modula-2 provides both a range-checked and an unchecked type conversion for those few occasions when you need to violate type rules. Modula-2 also provides the type ADDRESS, which can be used in arithmetic expressions and is compatible with any pointer type.

Though Modula-2 is based on Pascal, it retains few of the flaws of its ancestor. The article "Modula-2 - a Solution to Pascal's Problems" by Sumner and Gleaves in the September 1983

issue of *SIGPLAN* shows how Modula-2 systematically addresses the problems of Pascal as outlined by Brian Kernighan in the Bell Laboratories report, "Why Pascal Is Not My Favorite Language".

Unlike the Berkeley Pascal compiler, the DEC Modula-2 compiler on UNIX generates excellent code. With no optimizations performed, it emits object code that runs as fast or faster than optimized code from the Berkeley C compiler; in some benchmarks, Modula-2 runs in 70 percent of the time of the equivalent C program. With optimization, Modula-2 benchmarks run in 90 percent to as little as 33 percent of the time used by C. Even if all runtime checking is turned on, the optimized Modula-2 code is usually faster than C, which, of course, has no checking.

The Modula-2 compiler is fast, too. An internal version compiles about 1500 lpm when not generating runtime checks, and 1300 lpm when generating checks. Faster versions are expected as optimizations are applied to the compiler itself. (The compiler is compiled with runtime checks—the advantages of these checks outweigh the speed advantage gained by omitting them.)

Admittedly, Modula-2 is not perfect. For starters, it lacks C's structured initializers, Fortran's dynamic arrays and complex and double-precision types, and Ada's exception-handling mechanisms. Availability is limited: the DEC compiler is available to universities on 4.2BSD and Ultrix, and can only be had by those commercial organizations using Ultrix.

There are currently no textbooks on Modula-2, so the language is not yet widely taught. The UNIX compiler is now used more for systems programming research at universities than it is for instruction. This situation

SVS "C"

The SVS "C" language is now available on an expanded family of CPUs. It fully supports:

- Common "C" dialects

MC68000

- Ideal for Applications Development and Rehosting Programs for Improved Efficiency

- Emphasis on Floating Point

- Single Precision Option

- Integrated Hardware Floating

Point Interfaces:

MC68881, SKY, NS32081

- Interlinkable with SVS Pascal

- and SVS FORTRAN

- Symbolic Debugger

NS32000

- Optimizing Code Generation

- High Speed Compilation, No Assembler Passes

- Free of AT&T Licensing

MC68020

- Complete User Documentation

- Available since 1982

SVS has been a major OEM supplier of compilers since 1979.

For further information about these and other quality OEM compilers call 415/549-0535.



Silicon Valley Software, Inc

10011 N. Foothill Blvd., Suite 111
Cupertino, CA 95014

Circle No. 272 on Inquiry Card



LANGUAGE OVERVIEW

will change, though; I know of at least two textbooks to be published in the next year, and as the number of implementations on microcomputers grows, Modula-2 should experience the same sort of popularity explosion that launched Pascal.

In fact, many development and research organizations are already using Modula-2. At DEC, all new code being written at the Western Research Laboratory and Western Software Laboratory is in Modula-2, and DEC's Software Research Center is using an extended version of Modula-2. Other Modula-2 users include Bank of America, Bendix, Floating Point Systems, Ford, McDonnell-Douglas CSC, Phillips, Signetics, and Tektronix. (Ironically, some of these companies chose

Modula-2 over Ada on the recommendation of their defense-related subsidiaries.)

For all that, Modula-2 faces an uphill battle in the UNIX community. C is well entrenched, and programmers tend to be a conservative lot, often shunning the new because it is unfamiliar. But unlike any of the other languages that have preceded it, Modula-2 is designed to compete with C on its own turf—systems software. Modula-2 is more readable than C, more maintainable than C, and enforces consistency during system integration. Better still, a compiler that runs faster and generates better code than **cc** is available. I hope we will one day see copies of *Software Tools in Modula-2* by Kernighan and Plaugher on every UNIX program-

mer's bookshelf.

Joel McCormack co-authored the Z80/8080 UCSD Pascal interpreter while attending UCSD, where he obtained a Masters of Science degree. He then designed and micro-coded a 16-bit bit-slice board to execute UCSD Pascal for NCR, for whom he also wrote the high-level microcode language compiler. Compiling Pascal at 10,000 lpm hopelessly spoiled him. Later, at Volition Systems, Mr. McCormack co-authored the Modula-2 compiler, and eventually found himself President. After stock battles shut down Volition, DEC's Western Software Laboratory lured him away from San Diego's beaches to work on Mike Powell's Modula-2 compiler. He now programs in C only under protest. ■



**ACCESS
METHODS
INCORPORATED**

ATTENTION: UNIX™ SPECIALISTS

We have the most challenging and sought after consulting assignments in the UNIX* industry:

- Kernel Work
- Distributed UNIX™
- Networks (x.25 and LAN's)
- Graphics
- Real-Time Systems
- Languages, Compilers, and Translators
- Hardware and Microcoding
- C Applications Programming

See us at UNIX Expo, Booth 235
Drop your business card in our fishbowl for a chance at a free Video Recorder.

ask for us AMI

590 Valley Rd. ■ Upper Montclair, N.J. 07043
(201) 744-9126

314 West 56th St. ■ New York, N.Y. 10019
(212) 245-8114
call collect

*UNIX is a trademark of AT&T Bell Laboratories

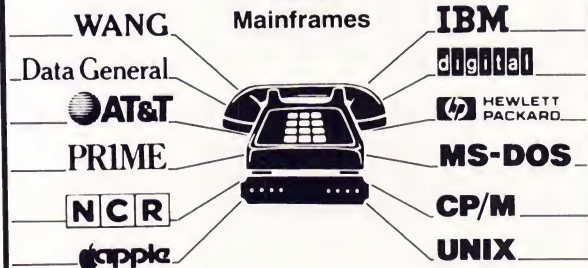
Circle No. 277 on Inquiry Card

BLAST®

Communications Software for

Micros
Minis

Mainframes



Any computer with BLAST can talk to any other computer with BLAST, the universal file transfer software linking many different computers, operating systems, and networks. No add-on boards; use any asynchronous modems or direct-connect for fast, error-free data transfer, even via noisy phone lines, satellites, LANS, and packet networks.

\$250/micros

\$495-895/minis

\$2495 up/mainframes

Communications Research Group 1-80024BLAST

8939 Jefferson Hwy. Baton Rouge, LA 70809 504-923-0888

Circle No. 276 on Inquiry Card

SERIX

ANNOUNCING
SYSTEM V
UNIX™

...puts your IBM Series/1® ahead of the pack!

SERIX is the high performance CMI version of AT&T's UNIX™ System V operating system with Berkeley 4.1 enhancements ported to the IBM Series/1 minicomputer.

SERIX transforms your Series/1 into an even more powerful, flexible, and convenient processor for general data processing, office automation, communications, and process control. Its advantages are outstanding:

Reduced software costs

Long term growth path

- Software is highly portable
- Provides access to a large, growing software base

More power from the Series/1

- Optimizing C compiler uses native code features
- All code reentrant
- Dynamic memory allocation without fixed partitions

Increased programmer productivity

- Large set of utilities
- Hierarchical file structure
- Pipes, forks, semaphores, and shared data segments

Other CMI Series/1 software

- RM/COBOL™
- UNIFY™ database management system
- ViewComp™ spreadsheet
- vi visual editor
- EDX™-to-SERIX™ conversion kit

CMI Corporation is a Master Value-added Remarketer of IBM Series/1 equipment. Leasing and other financial arrangements are available.

Contact us for further information.

Photographer - Michael Zagaris • UNIX is a trademark of Bell Laboratories
• SERIX is a trademark of CMI Corporation • SERIX was developed exclusively for CMI by COSI. • IBM, Series/1, and EDX are trademarks of International Business Machines Corporation • UNIFY is a trademark of North American Technology, Inc. • RM/COBOL is a trademark of Ryan-McFarland Corporation
• ViewComp is a trademark of Unicorp Software, Inc.

CMI 

A Torchmark Company

CMI Corporation
SERIX Marketing
2600 Telegraph
Bloomfield Hills, MI 48303-2026
(313) 456-0000

TWX: 810-232-1667
Telex: 499-4100 ANS: CMI CORP. BDHS

Member CDLA Member ASCD

The Firebreathers continue on the cutting edge of high performance computers.

The most powerful line of computer systems made. Gould PowerNodes™ and CONCEPT/32s®

Any way you slice it they beat the VAX.™

Our main-frame PN9000 and CONCEPT 32/97

are up to twice as fast as the VAX 8600.

And even though the mid-range PN6000 and CONCEPT 32/67 are 30-50% smaller than the VAX 11/780, they're still up to three times more powerful.

More power for a slice of the price.

Despite their superior power, our mid-



range models cost 40% less than the VAX 11/780. Our mainframes cost about 30% less than the new VAX 8600. The bottom line is more power for less money.

Operating environments that are a cut above the rest.

There's also a choice of system software to consider. Gould's unique UTX/32® is the first operating system to combine UNIX* System V with Berkeley BSD 4.2. So it allows you to access virtually any command format you want whenever you want.

And in real-time environments, Gould's MPX/32™ operating system offers performance that's unmatched in the industry, as well.

Delivery that's right on the mark.

Unlike the VAX 8600, that has up to a 12 month wait for delivery, when you

order either a Gould PowerNode or a CONCEPT/32 system, they'll be shipped within 90 days ARO.

You can also be sure with Gould you're getting a computer that's backed by years of experience – the kind of experience we used to develop the first 32-bit real-time computer.

If you need more information or just have a few questions, give us a call at 1-800-327-9716.

See for yourself why VAX no longer cuts it. Go with a Gould computer and ax the VAX.

CONCEPT/32 and UTX/32 are registered trademarks and PowerNode and MPX/32 are trademarks of Gould Inc. VAX is a trademark of Digital Equipment Corp. UNIX is a trademark of AT&T Bell Labs.

 **GOULD**
Electronics

Only Gould computers have a big enough edge to ax the VAX.



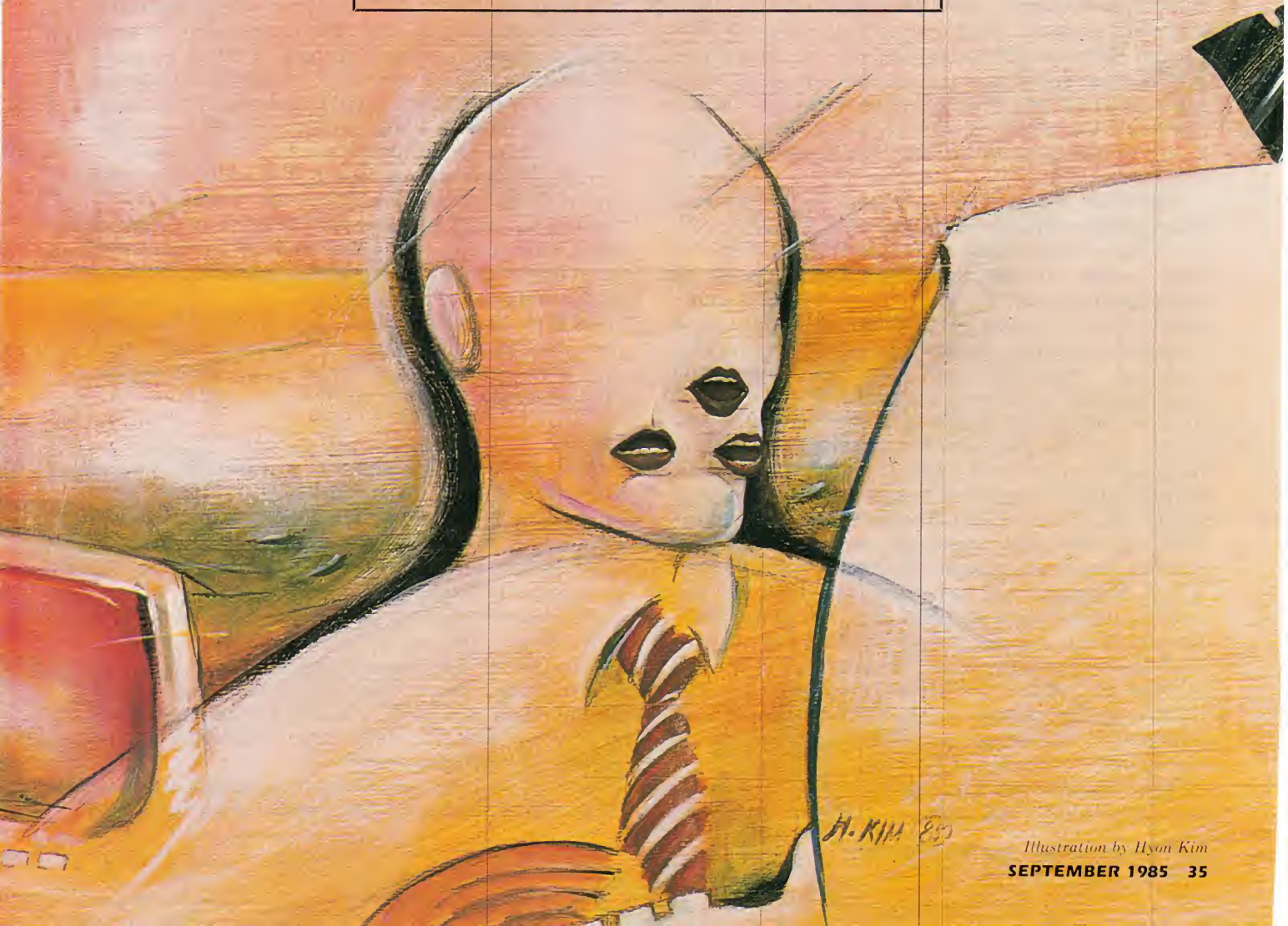
Circle No. 297 on Inquiry Card

THE HOUSE OF MANY TONGUES

Unto every purpose there is a language

by **Steve Johnson**

Rarely have an operating system and a language had such a symbiotic relationship as the one that ties the UNIX system to the C language. While C remains dominant, many other languages also have grown and flourished in the UNIX environment. Some of these are standard, but others are unique to the UNIX system. The "UNIX philosophy" of small, well crafted tools has encouraged mini-languages to be developed for every purpose. Also, some interesting tools have been popularized on the UNIX system that make the construction of new languages





The user brave enough to do text processing will quickly discover languages galore.

somewhat easier than on other systems.

This article will quickly tour some of the non-conventional languages found in daily use on UNIX systems, and then discuss the tools that have brought them into being. Finally, prospects for the future will be outlined.

QUICK TOUR

Taking an extreme and biased view, practically every program that interacts with a user has a command language. An experienced user of UNIX probably knows a large number of different languages. Some of these are small, others large. Some are regular, allowing a few simple elements to be combined in a large number of useful ways. Others are idiomatic, revealing little rhyme or reason. Let's take a look at a few of the languages available to the UNIX guru.

The shell. The text interpreted by the shell forms a language, with operators such as "&" and "|", grouping symbols such as "(" and ")", statements such as **for** and **do**, and expressions containing such symbols as "*" and "?", as well as more conventional names. Although the shell has a reasonably regular syntax, its semantics are very irregular; arguments may look the same from command to command, but generally they have different meanings. While this terrifies many new users, some have argued that the shell's command language is in fact well suited to its purpose (Harris, Marion, "UNIX Command Language", *Proceedings of the 10th Anniversary Usenix Conference*, June 1985, pp 343-348). Moreover, Jean Wood of DEC recently observed, tongue in cheek, that the command language has served as a force for the internationalization of UNIX systems since UNIX commands clearly aren't English. AT&T proposed a command syntax standard a couple of years ago (*Proposed UNIX Command Language Syntax*, UniForum Conference, January 1984) in an attempt to regularize the semantics, as well as the syntax of commands.

Text Processing. The user brave enough to do text processing will quickly discover languages

galore. The granddaddy of them all is **troff**, with roots reaching back to days far before UNIX, and a syntax and semantics that bespeak its original implementation in assembler. People usually process text using one or another macro package on top of **troff**, in effect defining other languages. Want equations, tables, pictures, graphs? Other specialized languages are available to do what you need.

*The **ed** editor and friends.* Text editors provide another rich source of input languages. The input language for **ed** allows for statements such as *a*, *s*, and *q*, and expressions such as:

```
/a.*e.*i.*o.*u/
```

Moreover, **ed** has been extended to provide a language for editing text streams, **sed**, and for use as a visual editor, **vi**. Regular expressions from it even surface in programs such as **grep**.

*The **awk** language.* The **awk** language can be used to handle character strings, pattern matches, and simple arithmetic. Recently, **awk** has been extended by allowing functions to be defined and invoked, making it look even more like a true programming language. Although it has been used for many years as a simple and powerful substitute for general database systems (especially for small applications), **awk** also is general enough that it has been used to write compilers (albeit slow ones!).

The C language. Of course, C is commonly used on the UNIX system. Enough said.

Conventional Languages. Various versions of UNIX offer one or more conventional languages in addition to C. On many UNIX systems, one can either find or buy Fortran, Pascal, Lisp, Modula, and several dialects of COBOL and BASIC. On some forward-looking systems, one can find Prolog, Ada, and C++. And don't forget that even simple languages like **bc** also can be quite useful.

TOOLS

How might the development and support of these languages be made easy? Obviously, by using more languages! The UNIX system supports a number of language-building languages as well as several tools that facilitate work on large projects—like the production of a language.

The earliest of these tools is **yacc**, one of the first application programs to run under UNIX (it's even older than C!). The **yacc** facility takes a description of a language (more formally, a LALR(1) grammar) and builds a C program called a *parser* that reads and structures the language, and detects and recovers from input errors.

As an example of how this might work, a language implementor might wish to restructure an

input language to allow integer expressions wherever only simple integers had been legal previously. This might be done as simply as adding the **yacc** rule:

```
integer : integer '+' integer
        | $$ = $1 + $3; |
```

to the existing **yacc** file. The first line can be interpreted as saying: "Wherever it is legal to input an integer, it is also legal to input an integer, a plus sign, and another integer." The second line, which looks a bit like a fragment of C code, says that the value of the integer on the left side (\$\$) is the sum of the values of the first and third components of the right side of the rule. Thus, users will be able to say "2+2" where they only could have said "four" before.

It is possible, though, that the plus operator is used in some way in the modified input language that conflicts with its use in integer expressions. The **yacc** facility will detect and flag those places where it is unable to decide which rule is to be applied. The language implementor then can eliminate such ambiguities before releasing the language to users. The **yacc** facility will also detect rules that can never be reached and other error conditions.

In addition to integers, **yacc** can handle more complicated values like pointers and structures. It is straightforward to generate parse trees or other data structures from input using **yacc** actions so that further processing can be done.

In most uses, **yacc** is concerned with the complex structure of the program. Lower level details are usually handled by another program known as a *lexical analyzer*. Lexical analyzers are typically used to recognize comments, blanks, constants, identifiers, multi-character operators, and the like. The individual characters in the input are collected and processed, and the parser is told what they represent. Lexical analyzers also keep track of file and line numbers, so that error messages can describe the location of discovered mistakes.

One tool, **lex**, can be used to build lexical analyzers. As with other UNIX tools, input to **lex** consists of patterns and actions. In this case, the patterns describe chunks of input text called *tokens*. As a token described by a pattern is recognized, input characters that match it are collected and the action specified by the user is performed. A sample **lex** line reads as follows:

```
"while" | return( WHILE ); |
```

This could be used to recognize the reserved word *while* in a programming language. When the word is located, the action returns a value to the parser

The UNIX command language style was a reaction to the chatty nature of some other operating systems.

indicating that the keyword has been encountered. A more complex **lex** line like:

```
[0-9]+ | yy1val = atoi(yytext); return( INTEGER ); |
```

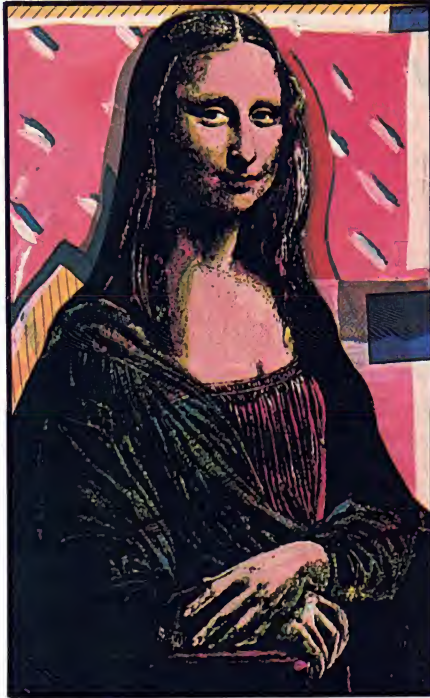
might be used to recognize integer constants in a programming language. The pattern matches one or more digits between 0 and 9 and specifies an action that is considerably more complex than the one in the first example. This action calls the library routine **atoi** on the array, *yytext*, where the integer has been collected. The resulting value is stored in a special variable, *yy1val*, where **yacc** can pick it up if desired. The action then returns to the parser an indication that an integer has been seen.

Beyond **lex** and **yacc**, language developers also enjoy many of the other advantages of the UNIX system. In particular, **make** and the shell are nearly essential to the production of high-quality languages. When programs such as **yacc** and **lex**, which produce other programs, are regularly invoked, a tool like **make** is essential to ensure that changes are reflected accurately in the recompilation process. This is particularly important since languages are often used by many users. Shell files that automatically do regression testing can help ensure high quality by checking to see that the latest version of a language has not broken any key applications.

Inputs for **yacc** and **lex** follow what is known as the *pattern/action* format, which is also used, with modifications, by **awk** and **make**. All these languages have rules that are evaluated, and actions that take place when the rules are satisfied. What makes the languages interesting, though, is that the flow of control depends on the patterns, rather than on more conventional control flow statements such as **if** and **while** statements. Each pattern/action pair tends to stand alone, and have meaning outside of the immediate context in which it appears. This may account for the observation that programmers

tend to reuse the **yacc**, **lex**, and **make** files from others more readily than the programs produced by others. Pattern/action languages appear to require a

Continued to Page 99



THE SOURCE CODE MAINTENANCE CHALLENGE

Tracking an ever-changing picture

by **Marc J. Rochkind**

Through development and first release, a new software product undergoes five distinct phases: specification, design, implementation, testing, and packaging. Only those products that are successful, though, enter yet another phase: maintenance. If a product is very successful, its maintenance phase will last many times longer than its various development phases. For example, the UNIX operating system entered its maintenance phase about 10 years ago, and it looks as though it will be maintained for many years to come.

Unlike the maintenance of tangible products (such as automobiles), software maintenance does not connote "repair or replacement due to wear and tear". Rather, to maintain software means to correct its design flaws

(discovered months or years after initial release) and make whatever modifications are necessary to adapt the software to new purposes or new environments. Automobiles are rarely modified to turn them into, say, pickup trucks, but analogous software transformations happen all the time.

Maintenance of tangible products is routinely done at the actual, end user level; manufacturing plans typically are not involved. By contrast, software maintenance generally does entail changes to manufacturing plans, not delivered executable programs. Users are then updated by replacing their executables with new ones; this is analogous to replacing an automobile when its ashtrays get full.

Because source code mainte-



nance is complicated by several factors not present during development, a seemingly simple change that might take only hours or days during development could take weeks or months during maintenance. I'll describe three major complicating factors.

COMPLICATIONS

The first, and most serious, complication is that original designs generally are partitioned into modules ("divide and conquer") according to specifications made at the time. These specifications change during maintenance, but modifications still are done within the bounds of the original partitions because maintenance programmers prefer tweaking code to totally reorganizing systems. The result is that software partitions, which are by far the most important property of any design, become less and less appropriate as time goes on.

A second factor is that systems get larger and fancier as features and more exotic performance algorithms are added. This is usually matched by growth in the maintenance staff, but because of turnover, the experience of that staff may be declining. The effect of this is that the quality of the programming decreases with time. Inappropriate partitioning accelerates this decay.

It seems obvious that at some point in the life of a software product a new set of specifications should be drafted, a new partitioning be created, and a new system be developed from scratch. In practice, though, this is almost never done. There are several reasons for this, not the least of which is that personnel may be unavailable. A product that has been out for some time is always more complex than the original, so the job of specifying and designing its replacement

will be that much harder. The development of a replacement may take two or three years, but by that time maintenance on the old product will have changed the specifications even more. Finally, the emotional upheaval caused by a completely new replacement may be more than customers can bear (consider new Coke vs. old Coke).

The third factor is that the presence of numerous versions of

Automobiles are rarely modified to turn them into, say, pickup trucks, but analogous software transformations happen all the time.

the various software modules, each in different stages of development and release, makes the task of manipulating source code confusing, error prone, and inefficient. Of the three complications I've described, this is the only one that has been satisfactorily handled in an automatic way. One tool that does the job is called the Source Code Control System (SCCS), first implemented under UNIX at Bell Laboratories in 1974. SCCS became an official part of UNIX with System III, and it is now available with most implementations of UNIX.

SCCS acts like a new member of the maintenance team: the librarian. The source code for each module is stored in a special file format under the protection of SCCS. A module can be retrieved

for study, for compilation, or for editing via the **get** command only. After a module is edited, the changes are entered formally via the **delta** command. There are a half-dozen or so other SCCS commands for changing access permissions, for listing changes, and for other administrative tasks.

When a module is first placed under SCCS control, normally at the end of development, it becomes the base for a new SCCS file. As changes are made, increments are added to this base by the **delta** command. These increments are called *deltas*. The programmer need not be concerned with the details of what the delta has affected unless he or she wants to. The **delta** command compares old and new files in their entirety and figures out for itself what actually has been changed.

With the **get** command, the source code for a module can be accessed at any delta. Effectively, this is the same as actually storing every version of every module in a separate file. But because of an encoding scheme used by **get** and **delta**, the additional space used to store the deltas is quite small relative to what would be required to save entire files.

Perhaps the key benefit of SCCS is that it records descriptive information about modules and changes to them that goes beyond what the UNIX file system itself records. For each delta, SCCS records who made the change (login name), when it was made, what source lines were changed, and even why it was changed (the programmer is prompted to enter a descriptive phrase). Deltas are assigned numbers designating a release (1, 2, ...) and a level within that release (1.1, 1.2, ...).

When a fix is made to a release, programmers who access that module at a later release are notified about the change so that

they can decide whether to include it as is, make an alternate fix, or skip it. This goes a long way toward preventing a bug eliminated during one release from creeping back into the next release, a slip that's all too common when versions are managed manually.

When SCCS was first deployed, some managers got the idea that information about deltas could be used for producing programmers' performance appraisals, but, happily, this hasn't caught on. In fact, some programmers make lots of deltas (one for each time they invoke the editor), while others make a delta only when they are completely finished with a set of modifications. There is absolutely no correlation between

the number of deltas on a module and the skill of the maintenance programmer or the reliability of that module. Indeed, given a choice of modules in which to insert a modification, the cleanest, most readable module is the one most likely to be changed.

An important reason for the success of SCCS is that it is philosophically neutral: it does not impose a policy detailing how a software product should be maintained, but rather simply helps each project—and sometimes each individual—automate according to the dictates of their own policy. Actually, most programmers who use SCCS never execute **get** or **delta** directly. Instead, they use shell procedures custom-designed especially

for their project. Besides executing **get** and **delta**, these procedures may perform other tasks such as sending mail to a project workbook login, kicking off compilations, generating reports, and so on. Thus, SCCS should be viewed not as a software administration system, but rather as a tool that can assist in creating such a system. The enormous flexibility of the UNIX shell and its associated software tools allows each project using SCCS to do things its own way.

Marc J. Rochkind invented the Source Code Control System in 1972. He is currently President of Rochkind Software Corporation, of Boulder, CO, which markets the RIDE programming language. ■

ASSEMBLERS

use **UNIX™** to put your ideas into ROM

68020 80286 Z80 64180 8051 6502 6809 8041 8048 8085 6800 6801 6805 68HC11 1802 Z8 7000 3870

UNIWARE™

- ★ well supported
- ★ well designed
- ★ well documented
- 8/16/32 Bit Macro Assemblers
- Link Editor (overlays)
- Debugging Aids
- EPROM Utilities
- ★ Open System
- ★ User Configurable
- ★ UNIX-driven

WE WILL SUPPORT YOU ON OVER 20 UNIX-BASED COMPUTERS WITH THE FOLLOWING GUARANTEE

Use UniWare for your project for 30 days. It will meet or exceed your expectations. Test UniWare – in any way you want. Go to the limit. And if you're not more than satisfied that it's shown you exceptional merit, send it back for a full refund. Only SDSI will give you such a broad guarantee. We support the chip you need on the system you want. Today.

SOFTWARE DEVELOPMENT SYSTEMS, INC.
3110 Woodcreek Dr. • Downers Grove, Illinois 60515
U.S.: (312) 971-8170 England: Unit-C, Ltd. (0903) 205233

Unix is a trademark of
AT&T Bell Laboratories

... with the **FINEST Software Support Service in the business**

UNIWARE is a trademark of
Software Development Systems, Inc.

Circle No. 273 on Inquiry Card

Of the several hundred charter members of the UNIX community, few are as well known as Stu Feldman. Long service at AT&T Bell Labs and a sabbatical at UC Berkeley put him in contact with almost all of the system's early developers. His strong opinions and brilliant wit, meanwhile, have made him a favored Usenix speaker.

Now District Research Manager of the Software Engineering Research Group at Bell Communications Research in Morristown, NJ, Feldman has also gained notoriety for his work on compilers and language tools under UNIX. Among the best known, of course, is the Fortran 77 compiler he developed (with Peter Weinberger). In addition, Feldman authored the EFL (Extended Fortran Language) pre-processor (which compatriots claim really stands for "Even Feldman Likes It") and **make**, the powerful program development tool that allows for the smooth integration of changes.

Based on this evidence, UNIX REVIEW felt that if one person could provide insights into the state of language technology, Feldman would be the one. Sure enough, when Dick Karpinski, manager of UNIX services at UC San Francisco, conducted the interview, he found no shortage of opinions.

REVIEW: You've probably had occasion to consider the state of compiler technology. Do you think that advances are necessary?

FELDMAN: That would appear pretty clear in a large number of directions. The compilers that most of us use aren't very good. They're slow, they don't generate very good code, they don't give especially good error diagnostics, they don't help you keep track of large programs, and they're written in ways that are very hard to understand. Almost everything you can think of is wrong—apart,



STATE OF THE ART

An interview with
Stu Feldman



of course, from the fact that we do use them and they do compile.

REVIEW: I didn't even hear you mention bugs.

FELDMAN: I was beginning with the assumption that we were discussing compilers that actually worked. There are several different classes of bugs. The important compilers—that is, the ones for pre-existing, old-fashioned type languages like C, Fortran, and you name it—have to compile languages that have details that don't actually make sense. So, the compilers often are written by people who have never understood what they were supposed to do in the first place. Sometimes there are bugs that reflect this misunderstanding. Then, of course, there are bugs that exist simply because languages are not easy to compile for.

It's very clear that compilers aren't nearly so satisfactory as they ought to be. If you read the textbooks, it sounds like anybody who has passed a junior-level class in compiler construction ought to be able to go out and write a reasonable compiler. Unfortunately, many of these people actually do go out and try to write one, which is a disaster.

REVIEW: Are new languages necessary? To what end?

FELDMAN: Yes, they are necessary. They will show up continually because they're needed a) to express new problems and b) to represent new ways of looking at old problems. Presumably, the generation that will follow C (although it's not clear who's going to win) will represent serious changes in what the world thinks it needs. To quote a friend, "The language changes have been so successful that we've come full circle back to Simula." This was meant as a jocular statement, but it reflects how concerns recycle and how new ones enter consideration. For this reason, I'm fairly certain that languages will continue to be constructed for very good purposes. Of course most of

the languages to come will be losers, both in the evolutionary sense and by measure against any aesthetic standards.

REVIEW: *Languages are sometimes very close to their predecessors. Take Algol-60 and Pascal, or Simula-67 and Modula-2, for example. Are the differences between these languages large enough to notice?*

FELDMAN: When seen from far enough away, yes, they're almost identical languages. Seen close up, the differences are truly enormous, and these represent the very different things that are being accented. You can begin by saying Algol-68 had everything, which was one of its major problems. PL/I was even more hideous aesthetically because some form of almost everything that anybody ever wanted was stuffed into it. Other languages, incidentally, are sometimes even worse.

The Pascals and the C's of this world have succeeded because they offered very strong "smaller-is-better" reactions to the earlier languages. They fit on tiny machines fairly nicely.

REVIEW: *Would the UNIX system have been successful without C?*

FELDMAN: Yes, I believe so. Remember that there were UNIX versions in assembler and B, preceding C. A system like UNIX requires a language *like* C to fly, but something other than C itself could have made UNIX a reasonable success.

I should say that I like C a lot, so any comments I make should be taken as the criticisms of a happy user. C matches a model of thought that says, "I really want to be able to control what happens on an ordinary computer, often at a fairly detailed level. Most of the time I want to talk at a higher level, but I still want to know that I'm programming a computer, not describing a mathematical object."

I have a definite model of the machine C runs on. It's a thing

I would assume that by now C has dug in sufficiently deep that we're going to have it the rest of this millennium at least.

containing cells that hold arrows and cells that hold letters—that sort of thing. I picture myself manipulating those cells. This is a very congenial view for me. It also turns out to be quite easy to map onto most of the hardware we care about. This means that it is easy to write a reasonably efficient C program, and that it's relatively easy to understand a C program that's been written well. The language occupies a middle ground and matches the image UNIX grew up with; it's simple and well suited to running on happy little machines.

REVIEW: *Do you think that the code in the UNIX kernel and the various system utilities offer examples of well-written C programs?*

FELDMAN: At this point I suspect it would be unreasonable to say that the kernel is an example of well written C. It contains excellent C programs, but of necessity they have been fiddled with to fit performance constraints and so forth. Because of that, I suspect that system software is not the place to look for elegant design no matter what system is under discussion.

REVIEW: *And yet, C is capable of being used in that way.*

FELDMAN: C is perfectly happy with being moderately abused. My code almost never has **gotos**. That's a religion I sort of accept. I'm suspicious whenever I see

gotos in other people's software and I certainly go out of my way to avoid them in my own programs because I find them irksome. There are plenty of places, however, where if you want to speed things up a tiny bit and you really know what you're doing, **gotos** can be appropriate. For that reason, I think you'd do better to look into ordinary commands if you're looking for examples of reasonable style, simply because the demands put on the commands are not as strange. I'm not saying that the kernel is written badly; it's just that the most interesting parts are likely to contain some obscure code that's been carefully thought out.

REVIEW: *How long is C going to be with us?*

FELDMAN: I would assume that by now C has dug in sufficiently deep that we're going to have it the rest of this millennium at least. I also fully expect to be getting computer mail on UNIX systems five years from now. UNIX and C models both have turned out to be so satisfactory as ways of thought that, given the inertia, I can't see either of them disappearing during the next 15 years.

REVIEW: *In the same way that the Sholes [QWERTY] keyboard has proved satisfactory for most of the century?*

FELDMAN: There are zillions of arguments about why there is almost nothing dumber than QWERTY, but luckily I haven't had to learn another keyboard.

REVIEW: *What are the limits of language technology?*

FELDMAN: The technologies of compilers have lots of limitations because we just don't understand how to solve all kinds of important problems. However, the factor that will determine both the success and limitations of a language is what surrounds it. An environment offers lots of tools, mechanisms, and customs.

Let's hark back to Fortran for a



moment. All manner of assumptions are made implicitly about what is and isn't good about Fortran. People assume that **do** loops are good and will be optimized. Therefore you contort your programs to use them, and hope you don't have to debug them later. You make assumptions about certain things in the compiler technology based on what has been handed down to you through folklore. In this instance, the reality is that many compilers aren't very good at **do** loops. Certainly mine isn't.

Take the UNIX environment surrounding C. As a user, I expect there to be many interesting tools that can operate on the intermediate product of the C compiler—either assembler intermediate for special cases or the *.o* object format. All that together with enormously useful libraries constitutes the world of the C program. The C language itself comes with at least an I/O library, and, in reality, we tend to assume all kinds of other goodies. That's part of the world that has to be included in any consideration of language technology.

REVIEW: *How are the language tools used under UNIX?*

FELDMAN: One answer is that they aren't used enough. UNIX is not a single-language system in any way. Most programs written in low-level language are written in C. This is indeed true. But it is possible on your standard UNIX system for Pascal, Fortran, and C procedures to call each other happily. I consider that a major property that the language tools should reflect. Secondly, a much more popular tool language than C is the shell, all three or four major versions of it. Many more programs in some sense get written in the shell than in C.

REVIEW: *The volume of code may not be as great.*

People simply don't realize how much they use language-based tools in the UNIX environment.

FELDMAN: No, but every time someone types an interesting sequence in the shell, they're programming. Many other tools are truly languages, like **awk**, **sed**, and certainly **grep**. These unquestionably have language components, and the people who use these components all the time grumble furiously. Certainly things like **eqn**, the equation typesetter, or the table formatter **tbl**, are languages unto themselves—although they represent very opposite technologies. There's no question but that they're language-based tools: **eqn** actually has a **yacc** grammar. The **tbl** formatter doesn't, though, because it has a very trivial format. Then there is **troff**, which is another language. That's why **eqn** and **tbl** were written—to cover up the incredible syntax and horrors of **troff**. All of them, though, represent uses of language technology. First, the construction of these tools very much reflects an approach to breaking things up into pieces and using other people's tools to manipulate them. Secondly, whenever I use these languages, I feel like I'm programming something in them. Certainly there are people who put together **sed** scripts to do really weird things without thinking that they're using language technology, whereas in reality they're using recognition technol-

ogy that's quite fancy. People simply don't realize how much they use language-based tools in the UNIX environment.

A rather different example is a silicon compiler I did a few years ago called *XI*. [See *Proceedings of the IEEE Conference on Computer Design, 1983*.] The syntax is very C-like, but the semantics, of course, are completely different. The first version used macros to generate function calls. Then I used an early version of C++ [by Bjarne Stroustrup] with classes. The last one was a modified version of the front-end of a C compiler. Steve Johnson worked on that one. Assignments created components on silicon and there was a lot of runtime checking to make sure the silicon design rules were satisfied.

REVIEW: *But have we painted ourselves into a technological corner in the UNIX system environment?*

FELDMAN: I don't think we're actually in a serious corner at this point. UNIX has definite limitations, but that relates to how it was modeled. The "small-is-beautiful, tools-are-independent" view is probably something that's going to be hard to shake. Whether some other viewpoint is actually going to be needed in the near future, or whether UNIX will adapt, it's true that many of the language techniques used today under UNIX were designed to get around the fact that UNIX processes communicate via an ASCII line. If there were some way to send packets that were self-describing or structured, perhaps there would be less parsing going on. That's not clear. The UNIX model of a single processor running on a modest machine with an elementary process structure may cause some problems. But at least it's a very comprehensible model that ap-



the problem well enough, C is frequently a convenient way to express it. Doug McIlroy has made an interesting distinction: C is what you use if you are writing a program to do something that you already know *how* to do. If, instead, you want to talk about software *qua* software, this is the domain of the languages of the late '70s—abstract data-based languages that offer ways of describing scientific problems and software issues. But then there are languages like Lisp that are great for doing things you didn't think you could do at all. Therefore, it doesn't matter how badly you do them. The original contributions to AI that were written in Lisp are truly astonishing.

REVIEW: *Are CLU and Alphard "languages of the late '70s"?*

FELDMAN: Yes, the various languages of that family are experimental because there is no great body of work outside of the home organization. They really were attempts to work out the implications of some of the ideas of data types, and languages. No language that has followed from that family has really gained acceptance. Ada comes to mind, but let's just say that I am not excited. It's an attempt to have the worst of both worlds.

REVIEW: *Does Modula-2 suffer from the same defects as Ada?*

FELDMAN: No. Modula-2 is a very plausible middle ground. I've not written any serious programs in it. I've read some programs, and I've talked to a number of people whose views I respect on this. Modula-2 is a very plausible contender for the sweepstakes. Although it permits you to write things at a reasonably abstract level, it still is a sufficiently simple language at a low enough level that you can understand what's going on if you write in a

pers to run nicely on a large number of machines. It's clear that UNIX starts with a limited base that only uses a small part of the parameter space available to it. For a while at least, this makes it possible for UNIX to be cheerfully extended; there are people who sell UNIX machines with 12 processors, you know.

REVIEW: *One view of pipes is that they are a specialization of the co-routine notion. Doesn't this accentuate the "small-is-beautiful, one-tool-to-a-purpose" view?*

FELDMAN: Pipes are not quite strong enough to make co-routines convenient. If you decide that you're interested in things like co-routines, then you want a different signaling mechanism than that offered by vanilla 6th Edition-ish UNIX. However, with all the IPC mechanisms that people have added, you can do message passing implementations of co-routines. But it's not easy to put together complicated plumbing to send weird signals back and forth. Both the signaling and the pipe mechanisms are a bit

restricted in what they're really good at. As you try to build more and more complicated things, you run into more and more restrictions. People who build transaction processing systems that need communications between lots of different processes find difficulties in setting this up. This is why the larger, more baroque systems of today have added communication mechanisms. These additions are attempts to get you out of a corner.

As you can see, I have contradictory views on the subject: on one hand, I fear that UNIX, having a very definite model of what it is supposed to be good for, will end up having problems handling the general case—because even though you can stretch things, you end up with stuff that is relatively monstrous. On the other hand, UNIX allows you to hack a solution for almost any problem, and that can buy you more time.

UNIX, and C in particular, is used by people who have to get work done—and got tired of doing it all by hand. So they wrote a program. When you understand



straightforward fashion.

One of the joys of—shudder—BASIC and Fortran is that with modest tasks, you can just start at the beginning and off you go. That's a hacker's dream, but if you have a program that's reasonably complex, you can handle it with a moderate amount of effort in Modula-2. Ada and the languages that it came from were not designed with that as a goal. Simple things just aren't simple. The only hope for those languages is that they might skirt some of the complications inherent in complex tasks.

REVIEW: *Steve Bourne claims that there is a genuine advance to be made if we can afford to use Lisp-like languages. But he wonders if we aren't already too committed to C technology.*

FELDMAN: This is a religious argument—of course you can always convert any C-type structure reference to a not necessarily efficient sequence of CDADRs and CDADADRs, and there are programs that will do that. Similarly, any C program can happily have arbitrarily complicated list structures. C lives on pointers. I fail to see an actual dichotomy.

What the real difference is, I believe, is that Lisp programmers believe they have *the* solution. C programmers are happy to believe they have *a* solution. It is very easy to write a C program that's not very efficient. I write a large number of them.

The major difference comes in the environment that surrounds the languages, the persistent workspace of a Lisp program as opposed to the evanescent one of a C program. This, I think, is the more important distinction. That's something that you can get around if you decide to take a different attitude toward the C environment. This is the very sort of thing that I find of research



I really do not believe there is one language that will combine them all and prove to be the right solution.

interest. It doesn't violate any of the canons of UNIX or C to consider changes like that.

REVIEW: *Is there a change coming in the roles of C and Lisp?*

FELDMAN: Not that I can see. First of all, let it be observed that many UNIX systems run both languages in the same address space. Franz Lisp can arrange to call C routines, which goes to show that the two can coexist, if not cheerfully then at least grumpily—or perhaps “conjurally” is fairer. I don't think there is any obvious change coming in that relationship. People who like Lisp will refuse to do anything else anyway. People who program in C will, on the other hand, find that

to be perfectly congenial—because it matches the way you get at a UNIX system and because it matches a way of thinking.

REVIEW: *Is there a need for language research? In what directions?*

FELDMAN: There is certainly a need for language progress. What that means for language research is a somewhat difficult question. The results of conscious language research have entered the canon of Computer Science without affecting work in any short-term way. Take the data type work of the mid-'70s, for instance; it's not quite clear whether Ada represents the nadir or zenith of that work.

For all that, there clearly are language directions that need to be investigated further. Everybody, of course, is charmed by languages in the VisiCalc image. That represents a fascinating approach toward language in that it gets people to use computer languages while making them think they're just getting the computer to do things for them. But I don't know anybody who has managed to make major progress there. Clearly, theoretical work being done with semantic underpinnings has been valuable. I've truly been impressed with demonstrations of the automated way in which things can be generated to interpret and execute a program based on a really deep semantic description. The fact that all such systems tend to run two to three orders of magnitude slower than you would expect is also a little depressing. Nevertheless, I'm assuming that the theorists working on semantics are eventually going to sharpen the understanding of what really does and doesn't belong in languages.

Quite different are the people who simply are interested in con-

NEW SCO RELEASES!
XENIX SYSTEM V
FOR PC XT - AND PC AT - NOW!

“A UNIX™ TO BE
PROUD OF!”

—Kaare Christian, PC Magazine



Mankind searched the world over
for the multiuser operating system of the future.

Then IBM® chose XENIX® for the PC AT. And the future was *now*.

THE SANTA CRUZ OPERATION PRESENTS

XENIX NOW!

AN SCO PRODUCTION IN EXCLUSIVE ASSOCIATION WITH MICROSOFT CORPORATION
THE MULTIUSER, MULTITASKING PC BLOCKBUSTER “XENIX NOW!”
STARRING VISUAL SHELL • MULTISCREEN™ • MICNET • THE BERKELEY ENHANCEMENTS

AND INTRODUCING **C-MERGE** AS THE MS-DOS DEVELOPMENT ENVIRONMENT

FEATURING WORLD FAMOUS SCO TRAINING AND SUPPORT FOR DEALERS • END USERS • ISVs • OEMs
AND AN INTERNATIONAL CAST OF HUNDREDS OF XENIX APPLICATIONS

INCLUDING **LYRIX™** AS THE UNIX/XENIX WORD PROCESSING SYSTEM

PRODUCED AND DIRECTED BY THE SANTA CRUZ OPERATION

SCREENPLAY ADAPTED BY THE SANTA CRUZ OPERATION FROM ORIGINAL STORIES BY MICROSOFT AND AT&T
IN BREATHTAKING SELECTABLE COLOR

NOMINATED FOR ★ BEST DOCUMENTATION! ★ BEST SUPPORT! ★ BEST TRAINING!
★ BEST ELECTRONIC MAIL AND NETWORKING! ★ MOST APPLICATIONS!
★ MOST COMPLETE UNIX SYSTEM!



RELEASED FOR MOST POPULAR PERSONAL COMPUTERS.
APPLICATIONS ALSO AVAILABLE: LYRIX, MULTIPLAN®, INFORMIX®,
LEVEL II COBOL™, 3270 MAINFRAME COMMUNICATIONS.

(408) 425-7222
TWX: 910-598-4510 SCO SACZ

Circle No. 228 on Inquiry Card

M MULTIUSER OPERATION SUGGESTED
XENIX WILL TURN YOUR PC INTO A REAL COMPUTER

©MCLXXXIV The Santa Cruz Operation, Inc.
The Santa Cruz Operation, Inc., 500 Chestnut Street, P.O. Box 1900, Santa Cruz, CA 95061 (408) 425-7222
UNIX is a trademark of AT&T Bell Laboratories • Lyrix and Multiscreen are trademarks of The Santa Cruz Operation, Inc. • IBM is a registered trademark of International Business Machines Corporation • XENIX and Multiplan are registered trademarks of Microsoft Corporation • Informix is a registered trademark of Relational Database Systems, Inc. • LEVEL II COBOL is a trademark of Micro Focus, Ltd.



structuring programs that do something useful. When you look at their programs, you'll find that they actually embody a language, and that they reflect the changes that have resulted from the move from the fixed-card-field command languages of the '60s to the interesting command syntaxes of the '70s, to the menu-driven or fill-in-the-box applications of the '80s. I view all of them as languages, and I view all of these activities as part of language research—even though most of the projects I've described could not be published in respectable language journals. That's really more of a condemnation of the language journals than it is of the work.

REVIEW: *How do you account for the success of C, given that better languages like Algol-68 were available at the time of C's introduction but were not accepted?*

FELDMAN: Although more comprehensive languages like Algol-68 existed at the time, they did not run very cheerfully on 64K PDP-11s. It's not clear to me that Algol-68 is actually a better language than C. It is more comprehensive, it has a neater intellectual base, but in spite of that, C turns out to be easier to program in. You can say that since we're back to big address-space machines that are now cheap, anybody can suddenly afford Algol-68 or its moral successor, Ada, but I'm not sure about that. I would feel comfortable, though, saying that the real success of C stems from the fact that it made it possible to write programs that fit nicely in a restricted environment. It's also true that C fits into UNIX in a very nice way, and that it allows somebody who thinks at a relatively low but intelligent level to write good programs that turn out to be very portable. It's

easy to write C programs that are portable, and yet efficient in a large range of environments.

REVIEW: *When you say "portable", what is it that you mean?*

FELDMAN: That the same C program will run on a very wide range of hardware—typically under the same UNIX operating system. In reality, though, many of the programs will run on systems that don't have UNIX at all but simply support the C library.

REVIEW: *Without change?*

FELDMAN: Frequently without any change. With a little bit of effort you can harden them to run on systems that are at least moderately different. Even in my Fortran compiler, which of course is machine-language-dependent, because it has code generation requirements, the amount of code that is specific is minuscule compared to the size of the compiler itself. Other programs like EFL and **make** are almost totally insensitive to the systems they run on.

Portability is not a simple topic. At one point, it appeared that portability was a well understood problem—that all you had to do was get your language right. It's becoming clearer, though, as we agree on what we want to transport, that there's more and more to be encompassed in the model that underlies a portable program. The reason the programs I described are portable is that they only want to manipulate rather simple objects and do rather simple computations involving character strings. If my programs had to be really portable and totally independent of problems like variations in floating point, I suddenly would have a very different problem. And if I wanted to deal with very complicated, environmentally sensitive issues like

the manipulation of a wide variety of screens, I would have something very hard to port. A satisfactory definition then, from a practical point of view, is that something is portable if it's a lot easier to move than to rewrite. Something is truly portable if it moves almost entirely by itself. It turns out that if you are relatively careful, C can be a very good vehicle for writing programs that are efficient across the very broad range of systems that take in a considerable historic run of operating systems.

REVIEW: *Have we made any progress in language technology since the mid-'60s? How far have we come since Simula-67?*

FELDMAN: We've come all over the map. Simula-67 was a significant progress that was only one small branch of what was happening in languages. But if you believe in that branch, then Simula-67 represents a somewhat crude understanding of how classes might operate. Many languages that have followed have made better use of those ideas. Look at CLU and Alphard, for instance. Those are research vehicles that contain similar concepts but also support some serious work. Smalltalk ideas are another intellectual outgrowth of Simula thought, but perhaps that's not entirely fair. All of those languages represent very definite progress towards unspecified goals.

The important assertion is that there *is* no answer. I really do not believe there is one language that will combine them all and prove to be the *right* solution. Because if you tried to combine all the approaches, you'd end up with a very large language that does a mediocre job of representing anything. I have no objection to there being lots of different languages, each of which has a different base

of what it really cares about. A functional language for research is fascinating even though I am pleased not to program in one.

REVIEW: *Well, indeed, Edsger Dijkstra claims that minds are ruined by exposure to the concepts of Fortran.*

FELDMAN: Well, mine's ruined. By that definition I do not necessarily consider the purist approach to be the one that gets you anything of practical value or even of lasting interest. I will very strongly disagree with the view that one should only think clean thoughts without remembering anything.

REVIEW: *You think maybe that good minds can survive exposure to even ill-formed concepts?*

FELDMAN: Yes. I would say that an education that permits you to see more than one view of the world is probably superior to religion. My mind was spoiled at a fairly early age by learning machine language and Fortran, and also by learning Lisp and PL/I.

REVIEW: *How will the computing community address language technology limitations?*

FELDMAN: Most of the computing community in one sense will use whatever it finds convenient. Again, it harks back to the question of why C has been such a success. One reason is that it gives you very good access at a low level to the way UNIX runs. Therefore, it is the most convenient language, though not the only language, to use for programming in the UNIX environment. Therefore, anybody who is going to be a serious user of UNIX is probably going to use C at some point. Likewise, somebody who sits down in front of a Symbolics machine and starts using expert systems is going to end up learn-

ing Lisp at some point. The average person who simply wants to do something rather than talk about it is going to make the best use of whatever fits into the environment.

So the question is: what will research provide that will make life seriously better? In particular, how will research address the limitations of language technology? I believe that there will be some relaxation because systems are going to get considerably more comprehensive and smarter about what people are doing. "Programming environment" is a buzzword that doesn't carry quite the force that's required. But the integration of a large number of functions so that I can count on

my system to follow what I'm doing and help me out is probably more important than how I express what I want in a computer language. The Lisp systems at the moment are probably the most advanced in this way. They provide all kinds of approaches that make it possible to get out of trouble quickly, whereas in C or Pascal—C especially—you always get yourself in trouble and need help getting out. Simply inventing new languages isn't going to solve any of these problems. A better understanding of how people write software and how to let them write it better is something that's truly interesting. That's the direction I think development will take. ■

UNIX/'C'

Chance Of A Lifetime Project

The Brandon Consulting Group, one of the few firms offering a complete line of services in the field of information processing, from planning through implementation, has been awarded a contract with a major New Jersey based conglomerate producing a new line of micro computers.

As a Brandon employee, if chosen for this exciting project, you will be involved in the development of custom and packaged software for use on this new micro. In addition, you will be a member of a support team receiving "hot line" calls from all over the United States. You will be simulating problems that have been encountered and will be providing your expertise in solving them. This is a highly visible position.

The ideal candidate will have a minimum of 2 years current experience with UNIX/'C'. Any hardware environment will be acceptable.

We offer very competitive salaries and excellent benefits including three weeks vacation. For more information about us, or to apply, contact: **Natalie Fornal**.

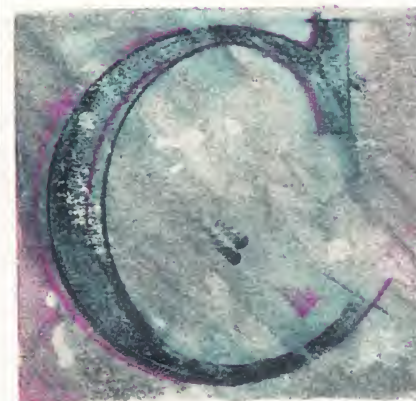


BRANDON CONSULTING GROUP, INC.

Corporate Headquarters

1775 Broadway, New York, NY 10019
(212) 977-4400
(201) 745-4700

Circle No. 256 on Inquiry Card



TOWARD ANSI C

How the standards
are taking shape

by **Thomas Plum**

The American National Standards Institute (ANSI) develops standards for practically everything—including the C language. ANSI Committee X3 bears responsibility for Information Processing Systems, and support for its Secretariat is provided by CBEMA, the Computer and Business Equipment Manufacturers Association. Operating within this framework is Technical Committee X3J11, the group that considers C standards.

The first meeting of X3J11 took place in June, 1983. Since then, it has held quarterly meetings across the United States. There are currently about 100 full members (or alternates) and a similar number of observers. In March, 1985, the committee voted to request ANSI X3 to distribute its current draft of standards as an *ANSI Information Bulletin*. This document is now available for informal public comment. (See the section titled "Further Information" later in this article for word about how to order a copy.)

The current schedule calls for publication of the draft proposed American National Standard for C some time after March, 1986. If this formal public review turns up no big surprises, we could expect to see an ANSI standard for C sometime late in 1986.

GUIDING PRINCIPLES

There are a few basic principles that have guided the committee's work. One of the most important is: *Don't Break Working Code*. User programs that conform to the Kernighan and Ritchie (K&R) description of C should continue to compile and execute as before. The same is basically true of programs written in more recent variants of C, except where some of the later variants have disagreed with each other. This means that the committee has adopted a "practical" point of view (in consideration of the large user investment in existing C programs).

Continued to Page 58

Illustration by Nancy Jorgensen

Since its introduction in 1959, Lisp has been the dominant language in the field of artificial intelligence. Of late, it also has become prominent in AI offshoot work done on expert systems. The spread of Lisp outside of academic circles is a fairly recent development, however.

John McCarthy, the author of the language, first unveiled it in the academic community shortly after the introduction of Fortran, and there for the most part Lisp has remained—save a few excursions into major research labs.

The large memory requirements and CPU demands of programs written in Lisp have limited their use in more general environments. Recently, though, radical improvements in hardware have opened new doors to the language. Lisp systems of one type or another are now available on a wide range of machines, including UNIX-based multiuser computers, customized Lisp machines, and small personal computers like the IBM PC and the Apple Macintosh. It seems likely that Lisp will continue to thrive, and that its use will grow exponentially as the increased availability and affordability of hardware allow the capabilities of the language to address a wider range of potential applications.

THE LISP STORY

Before considering how this growth will occur, though, let's first look at *what* Lisp is and *why* it's the preferred language in the AI field. Perhaps the primary reason is that a Lisp program is naturally represented in Lisp data structures. Thus, programs and data can be treated somewhat interchangeably, allowing Lisp developers to write programs capable of running and modifying other programs. This ability can be focused internally, allowing a program to make changes to lines of its own code while running. Because of this self-modifying ability, Lisp programs seem to "learn". For example, an expert system program incorporating a

LISP ON THE MOVE

How UNIX plays
a role

by Joel Hass





collection of rules that tell it how to react in various circumstances might discard some of those rules if it finds them not being used. It could then generate replacements. This involves taking specific examples and generalizing them into a useful rule. The fact that Lisp programs and data are similar make the Lisp environment much more convenient for this than the environments provided by such languages as Fortran. In practice, self-modifying programs are prone to disastrous results, so most AI programming is done using higher level concepts built on top of Lisp (OPS-5 offers a well known example). These specialized AI environments force additional discipline on program behavior.

Other useful Lisp features include powerful debugging facilities; the availability of both an interpreter for program development and a compiler for the production of fast code (thus allowing parts of a program to be run compiled while other parts are run interpreted); runtime type checking; dynamic storage allocation (known as "garbage collection" in Lisp parlance); and a macro facility that allows for easy extensions of the language.

The best Lisp engines of the 1960s and early 1970s were found in the DEC line of PDP-6, PDP-10, and PDP-20 computers. These machines came equipped with an excellent instruction set, a comfortable operating system or two, and what for the time was an enormous address space (2.5 MB). Running on these machines was a dialect of Lisp called MacLisp, which had been developed at MIT.

Some dialects of Lisp ran under UNIX on PDP-11s during this time, but UNIX and Lisp made their first significant link via a program named MACSYMA on PDP-10s. Containing over

One problem is that the name "Common Lisp" can be applied to any product by any vendor.

300,000 lines of Lisp code, MACSYMA made it possible to perform a large number of symbolic mathematical operations (such as differentiation and integration), cancellation of terms in fractions, and various types of differential equations.

MACSYMA was developed in MacLisp at MIT, where, among others, Professor Richard Fateman was involved. Soon after leaving MIT for UC Berkeley in 1974, Fateman turned his efforts to looking for a vehicle on which to run MACSYMA. Berkeley had no PDP-10s, but it did have a collection of PDP-11s running an early version of what was to become Berkeley UNIX. The UNIX operating system had already become the choice of many of the computer scientists at Berkeley, so when the MACSYMA group formed by Fateman purchased a VAX (with help from the National Science Foundation), it made arrangements to use UNIX on the machine.

Version 32/V, implemented by Tom London and John Reiser of Bell Labs, was the first variation of UNIX the group tried, but it failed to take advantage of the virtual memory capabilities of the VAX. Without the use of its virtual memory, the VAX actually offered less capacity than its predecessors. To correct this deficiency, Fateman and others worked to modify the system. The

end result was the release of 3BSD and Franz Lisp, an adaptation of MacLisp for the new system developed by John Foderaro and others under Fateman's direction.

By taking advantage of the VAX's virtual memory capabilities, 3BSD transformed the machine into a powerful Lisp vehicle capable of handling 20 users simultaneously. The adapted version of MACSYMA was renamed "Vaxima", and Franz Lisp was included with the various Berkeley UNIX distributions. In this way, UNIX has served to spread Lisp far beyond its academic womb.

LISP TAXONOMY

There are two main families of Lisp used by serious developers today. One, based on the MacLisp dialect, was developed on PDP-10s at MIT. Its descendents include ZetaLisp, Common Lisp, and Franz Lisp. The other major strain is the InterLisp family, which first gained widespread use on PDP-10s and now runs on Xerox Lisp machines. Dialects with smaller followings also exist, including PSL (Portable Standard Lisp), an early attempt at standardization that failed due to lack of power and portability; LeLisp, a French variant; Scheme; and T. The last two, which like MacLisp originated at MIT, rationalized certain aspects of functions and data typing. They also substituted more reasonable terminology for Lisp terms like *car*, which is used to refer to the first element of a list (in Scheme and T, the term is *first*). Obscure terminology such as *car* and *cdr* owes to Lisp's age. For instance, *car* is an acronym for "contents of address register", a reference to address fields in an early IBM 7090 series machine. Rationalizations of such acronyms might seem appropriate but Scheme

and T have made few inroads into a Lisp community grown fond of its *cars* and *cdrs*.

The Lisp world lacks a body that sets standards for it in the way the American National Standards Institute has set criteria for languages such as Fortran, so dialects that diverge from the major strains of Lisp regularly emerge. Because of this, a group of people from academia, industry, and government met in 1981 to establish a standard called "Common Lisp". A chief advocate of the Common Lisp effort was the Defense Advanced Research Products Agency (DARPA), the organization that funds the bulk of artificial intelligence research in the United States. DARPA was intent on consolidating Lisp development and artificial intelligence research so that results from the two projects could be shared. This was especially important to DARPA in view of the emergence of AI technology in robotics, speech and image understanding, and other applications of potential use to the military and industry.

The Common Lisp effort was somewhat controversial since it lacked input from such groups as the large InterLisp community using Xerox machines. Some, in fact, branded the effort as an "International Common Lisp Conspiracy". Nevertheless, the Common Lisp effort forged on, forming a large committee of volunteers for the discussion of issues. After some argument and dialogue, much of it via ARPANET mail, a small subset of the committee hammered out a final report. The specifications on which they agreed appear in a book written by G.L. Steele, *Common Lisp, the Language* (Digital Press), which appeared in May, 1984. Actual implementations of Common Lisp have recently surfaced, although none as yet have

UNIX is seen as an ideal delivery system for software produced on customized Lisp development hardware.

contained all of the Common Lisp features.

Common Lisp is a very large language, as one might expect of a product generated by committee. It includes many data types and functions not found in previously prevalent Lisp implementations. For example, complex numbers, rational numbers, and four types of floating point numbers and their associated functions are included. In part, this stems from a desire to make Lisp a full language, suitable for numerical computation as well as for symbol manipulation.

The main thrust of the Common Lisp effort, however, was to achieve a standard. Many people in the Lisp community felt that the lack of a common banner was scaring industry away. Perhaps they were right. Current indicators show that most Lisp users and developers are indeed embracing Common Lisp.

Does this mean Common Lisp will bring order, stability, and compatibility to the Lisp community? Not likely. One problem is that the name "Common Lisp" can be applied to any product by any vendor, leading to Common Lisp products that implement only a small fraction of the total language, and may or may not follow the specifications outlined in Steele's book in even those

small fractions. Some of the "Common Lisp" products currently available for the IBM PC and other micros are a good example of this phenomenon, which also is occurring on larger machines. A key phrase to watch for in descriptions of Lisp dialects is: "extended subset of Common Lisp". With some imagination, this description could be applied to a washing machine.

Beyond the problem of enforcing correct usage of the term, the Common Lisp community also must deal with the fact that many features were left unspecified by the people who took part in the effort to establish a standard. Debugging features were among those left out, as were editors, graphics interfaces, interfaces to other languages, mouse and window support, error handling, and commonly used language features such as *Flavors*, which is used for object-oriented programming. Since various implementations will incorporate these features in various ways, they are likely to be incompatible, despite their common "Common" label.

THE UNIX ROLE

How does the commercial AI community view UNIX? Currently, it's seen as an ideal delivery system for software produced on customized Lisp development hardware. High-end Lisp development machines simply are too expensive and too limited in function to be suitable for the distribution of expert systems to a wide range of sites. Personal computers, on the other hand, are widely prevalent but have too many limitations to run a full-fledged Lisp dialect. Thus, UNIX machines, which are already widely distributed for other purposes and offer per-user costs that are rapidly approaching the nominal point, seem to be the ideal solution.

fo · rum, n. (pl. FORUMS)

1. A public meeting place for open discussion. 2. A medium (as a newspaper) of open discussion or expression of ideas. 3. A public meeting or lecture involving audience discussion. 4. A program involving discussion of a problem by several authorities.



*eForum designed by Marcus Watts, Copyright 1984, Network Technologies International, Inc. (NETI).

Electronic meetings continue the automation of knowledge transfer which started with electronic mail.

Electronic meetings are an extension of the communications revolution which started with electronic mail. It takes seconds to send a letter using electronic mail instead of days via regular mail. Certainly e-mail is a giant step in automating correspondence between two people.

eForum goes yet further to provide immediate communications automation. But for groups. It creates electronic meetings which allow attendees to participate in discussions using the dynamic ebb and flow of points, counterpoints, comments and

conclusions just like in-person meetings.

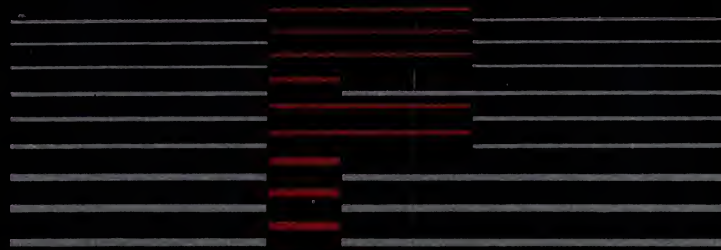
From an economics point-of-view, eForum is the most cost effective method for bringing together the best minds in your company to meet on key issues—without the price of a single plane trip, the aggravation of schedule conflict or time-consuming delay.

eForum is a communications breakthrough product.

eForum lets you create electronic meetings with attendee lists as large as the company staff or as small as a three-person design team.

Not only can eForum handle hundreds of meetings for your company, but, at the same time, limits each participant to only attending meetings to which he belongs.

eForum, n. 1. Low cost electronic meeting system (as in needing no scheduling or travel to attend). v. 1. Automatically organizes, indexes, files and leaves a complete written record of entire meeting. 2. Allows adding more attendees than normal at no extra cost. 3. Gives plenty of time to think before responding. adj. 1. Keeps everyone up-to-date. 2. Doesn't let geographic or time zones determine who can attend the meeting.



e **Forum**

The Electronic Meeting Manager

If you have ever attended a meeting,
you know how to use eForum.

Simply attend eForum meetings any time convenient for you. Review new discussion materials. eForum keeps track of what you've seen. Enter your comments or new discussion points. Instantaneously, your ideas are available to every member of your eForum group regardless of geographic location. That's productivity.

eForum has the flexibility to fit your communications needs.

- eForum 4000 - a national communications network available with a local phone call from most locations.
- eForum 2000 - UNIX™ based central host software for supermicro and minicomputers.
- eForum WS - software for the IBM PC and

compatibles to interact with eForum central host software.

Call 1-800-638-4832 or in Michigan call (313) 994-4030 collect for information on:

- Automating your company's meetings by using General Electric Information Service, the world's largest communications network, to tie together your microcomputers and terminals.
- Creating your own meeting network for your department or company. Software, hardware and leasing available.
- Establishing OEM and VAR agreements to enhance the value of your software or hardware, with the communications power of eForum. **Circle No. 285 on Inquiry Card**



**Network Technologies
International, Inc.**
The Arbor Atrium Building
315 West Huron
Ann Arbor, Michigan 48103

™UNIX is a trademark of AT&T Bell Laboratories
™eForum is a trademark of Network Technologies International, Inc.
(NETI)



As an example, consider a developer of stock options analysis systems at work on a Symbolics 3600 to produce a sophisticated AI program. After investing several staff years of work into the transfer of hunches from a prediction scheme to Lisp code, the developer understandably wants to deliver the completed expert system to hundreds of scattered financial analysts. These people have neither the need nor the desire for a Lisp machine that the developer's own AI guru has, so the developer naturally looks to the UNIX systems already located in many potential customer sites for a solution. In this scenario, UNIX gets the delivery pie while the manufacturers of Lisp machines reap the benefits of development work in the rapidly expanding AI market. UNIX programmers, meanwhile, have access to the object-oriented programming capabilities of *Flavors* and the functional programming abilities of languages such as FP (see the *4.2BSD UNIX Programmer's Manual*).

WHITHER LISP?

The prevalent opinion is that the Lisp machine market will explode. Arthur D. Little projects a Lisp machine sales volume of \$10 billion by 1990 (*Computer-World*, May 6, 1985, Update/7). Some developers, however, are finding that though 68000s or VAXen running Lisp may not have all the whistles and bells of Lisp machines, they can be used with a minimal (often invisible) investment of time and money. UNIX machines already can be found in many of the laboratories starting work on Lisp-based programs and expert systems. Lisp systems that can run on these computers are available at a relatively small expense. Many

Though 68000s or VAXen running Lisp may not have all the whistles and bells of Lisp machines, they can be used with a minimal (often invisible) investment of time and money.

manufacturers of 4.2-based computers, in fact, include a Lisp system as part of their basic offering.

For many, the faster speed of Lisp machines must be weighed against the extra effort and time it takes to acquire, install, and start using one. A good proportion of Lisp development work today is being done on VAXen, Suns, and Apollos purchased primarily for other purposes.

What of the case, though, where money is no object and the best must be obtained at any cost? Here, at least, where the scarcest resource is human Lisp expertise, can the Lisp machine stay unchallenged as the machine of preference? Perhaps not. A battle is looming on the hardware front for the hearts and minds (and wallets) of AI and expert system developers. On one side are the Lisp machine manufacturers—LMI, Symbolics, Texas Instruments, and Xerox—whose machines contain custom processors with specialized architectures designed to run Lisp. For the last few years, these

manufacturers have more than doubled their combined sales annually. Sales projections call for no abatement.

On the other side are UNIX system manufacturers who have targeted the same markets with machines based mainly on the Motorola 68000 and National Semiconductor 32000 families of chips. This camp also includes a scattering of additional machines such as the MicroVAX II from DEC. These UNIX machines have the advantage of being both less expensive and more versatile, with the ability to run a large variety of non-Lisp as well as Lisp software. Lisp machines currently sell at \$40,000 and up for single-user configurations, enough to buy a very respectable UNIX supermicro affording multiuser access. Prices for Lisp machines, of course, can run much higher—easily over \$100,000 with various options thrown in—though rumor has it that a \$10,000 version will be available by Fall '85.

There's no question but that the Lisp machines have a significant speed advantage. On standard Lisp benchmarks they perform from two to four times faster than 68010-based systems. Early evidence, though, seems to indicate that much of this speed advantage will disappear once the 68020 chip replaces the 68000 and 68010. The 68020 seems to run Lisp from two to four times faster than the 68010. Similar speed increases will come with the National Semiconductor 32032.

Apart from speed, another advantage currently enjoyed by Lisp machines is an integrated environment for Lisp development. Residing in this environment are a full array of bitmapped graphics, windows, debugging facilities, and object-oriented programming capabilities. These

features can and are being implemented on UNIX-based systems, however. When in place, these systems may strike a fatal blow to Lisp machines, or at least cause a precipitous drop in their pricing.

What can we expect to see of Lisp in the UNIX community in the meantime? First, we can look for continuing improvements in one or two of the Lisp dialects already present in the UNIX environment, while we can expect to see support for the other dialects dissipate at all but a dwindling number of sites. As an extension of this trend, the use of Lisp under esoteric operating systems will quickly disappear. The surviving dialects will flourish as their base of applications continues to grow, and as added features give them Common Lisp compatibility and allow them to take advantage of the faster processors becoming available. Along with the price of UNIX hardware, the prices of these Lisp products should drop steadily to the levels of current personal computer languages.

Second, a number of new Common Lisp dialects will appear that are likely to be incompatible with one another in many respects. DEC has already released a version of Lisp for VMS. Similar products have appeared—or will shortly—from Data General, HP, and others. In another three years or so, we should begin to see the arrival of parallel processor Lisp products designed to take advantage of the multiprocessor architectures of UNIX machines like those offered by Sequent and ELXSI. DARPA is already funding research that will bring this about.

Designing versions of Lisp that can take advantage of new technology in this way is one of the leading research projects underway in universities today. Hence, in a short time, we can expect to see Lisp co-processors that can be

plugged into systems much like floating point processors can today—at prices that also are quite similar.

Joel Hass is one of the founders of Franz Inc., which sells both Franz

Lisp and a new Common Lisp product, ExCL Common Lisp. He has a Ph.D. in Mathematics from UC Berkeley. Questions for Mr. Hass can be sent to 1141 Harbor Bay Parkway, Alameda, CA 94501 (415/769-5656). ■

Your UNIX* Course Source

FALL CURRICULUM SCHEDULE

UNIX Operating System	DATE
UNIX Operating System, a Conceptual Overview	Sep 3-5/Nov 25-27
UNIX Operating System for End Users	Sep 3-6
UNIX Operating System for Appl. Dev.	Sep 9-13
Advanced UNIX Commands	Sep 9-13
C-Shell Programming	Sep 9-13
Bourne Shell Programming	Oct 7-11
UNIX Systems Administration	Sep. 16-20
Advanced UNIX Systems Administration	Sep 23-27
UNIX Communications	Sep 30-Oct 4
UNIX Networking	Oct 7-11
UNIX Security	Oct 14-18
UNIX PROGRAMMING	
Basic C Language Programming	Oct 21-25
Intermediate C Language	Oct 28-Nov 1
Advanced C Language	Nov 4-8
SMC BASIC	Nov 4-8
INFORMIX/C Interface	Sep 9-13/Nov 11-15
UNIX APPLICATION TOOLS	
INFORMIX Applications Development	Nov 18-22/Dec 16-20
MULTIPLAN	Nov 18-22
UNIX/INFORMIX for End Users	Dec 2-6

All courses are taught at our training facility in Cincinnati, Ohio. The tuition is \$600 per week, except Conceptual Overview is \$400.

Our courses can, by special arrangement, be presented at your site — we can even bring the hardware!

Call now for Course Catalog and registration information: (513) 733-4747



Information Technology
Development Corporation

4000 Executive Park Drive / Suite 310
Cincinnati, Ohio 45241

*UNIX is a registered trademark of AT&T Bell Laboratories.

Circle No. 274 on Inquiry Card



C STANDARDS

Continued from Page 50

On the other hand: *All Implementations Will Change Somewhat*. From the start, it was agreed that no existing compiler would be held up as a model of perfection. This has meant that all implementers must accept a certain number of tradeoffs for the good of the process—a fact that has contributed to a relatively harmonious outlook.

Not All C is Written by Humans. C should remain suitable for use as an intermediate language. This has prompted the retention of certain syntactic forms that would be unlikely

Most X3J11 members feel that C does not need any "fixing".

to be produced by human programmers.

Coexist with Current Tools. The committee has avoided all changes to C that might require more powerful support tools (like linkers). The most hotly-debated proposal of this sort would have

required longer external name significance (31 characters with two cases, for instance, since the draft provides for internal names). After much discussion, the committee defeated the proposal upon deciding that primitive six-character one-case linkers could still be a part of a conforming implementation.

Another accommodation made to current linkers involves *common* linkage. The UNIX linker allows several source files each to contain an external declaration like "**int i;**", and be linked via *common*. Not all linkers can support *common* adequately for C, however, so the draft maintains the linkage restriction from K&R: all but one of the files must specify the word **extern** on the declarations. The UNIX *common* linkage thus becomes a system-specific extension, making it something programmers wanting to port to non-UNIX systems will need to avoid.

The committee also wants to permit cross-language linkage (such as the calling of C functions from Fortran programs), so no changes were adopted that would require unusual calling-sequence protocols.

Stay Close to the Machine. The committee feels it is vitally important that efficient code generation be allowed. For example, **char** still can be signed or unsigned according to the machine's characteristics. (However, ANSI C also will allow explicit specification of **signed char** and **unsigned char** in the interest of greater portability.)

Staying close to the machine also means keeping C available for machine-dependent uses such as device drivers. Such code is obviously and intentionally non-portable.

Allow System-Dependent C. The standard similarly avoids any deprecation of system-depen-

The Best 68000 & 32000 Compilers

GUARANTEED

OEMs: You get an unconditional 30 day money back guarantee that our compilers generate faster and smaller code than any other corresponding 68000 or 32000 industry standard compiler.

All compilers include 1 year of maintenance and updates.

C — FORTRAN 77 — Pascal

Available NOW for:

Motorola 68000, National 32000

UNIX 4.2 BSD, UNIX System V



Green Hills Software

P.O. Box 91030 • Pasadena, CA 91109 • (818) 796-6543

Leaders in stamping out vaporware and puffware.

UNIX is a trademark of AT&T Bell Laboratories

Circle No. 267 on Inquiry Card

NAME THE MOST WIDELY USED INTEGRATED OFFICE AUTOMATION SOFTWARE FOR UNIX™ SYSTEMS.

"UNIPLEX II"™

YOU'VE GOT IT!

User satisfaction is the primary reason no other product can make this claim. Already in its second generation, UNIPLEX II offers features designed to meet the requirements of the most demanding user.

The beauty of UNIPLEX II is its simplicity. One personality and one command structure throughout the program provide an ease of use never before experienced with UNIX application software.

UNIPLEX II integrates sophisticated word processing, spreadsheet, and relational database applications into a powerful one-product solution.

UNIPLEX II uses termcap, so it can run on virtually any computer terminal. "Softkeys" allow the user to define function keys which are displayed on the 25th line of most terminals to provide versatility and ease of use.

All this at a price you'd normally pay for a single application software package.

UNIPLEX II is available immediately from UniPress Software, the company that's been at the forefront of quality UNIX software products longer than anyone else.

Call today! Once you've got it, you'll see why UNIPLEX II is the most widely used integrated office automation software for UNIX-based systems.

OEM terms available. Mastercard and Visa accepted!

Write to: UniPress Software, 2025 Lincoln Hwy., Edison, NJ 08817 or call: 1-800-222-0550 (outside NJ) or 201-985-8000 (in NJ); Telex: 709418. European Distributor: Modulator SA, Switzerland 41 31 59 22 22, Telex: 911859.

UNIX is a trademark of AT&T Bell Laboratories. Uniplex II is a trademark of Uniplex Integration Systems.

*NOW AVAILABLE ON THE
AT&T UNIX PC T300
& 3B SERIES!*

UniPress Software
Your Leading Source for UNIX™ Software

Circle No. 291 on Inquiry Card

See us at UNIX EXPO, New York, Booth #300

C PROGRAMMERS' DBMS



db_VISTA

PREFERRED
over ISAM
and file utilities,
POWER
like a mainframe

DBMS, PRICE like a
microcomputer utility,
PORTABILITY like only
C provides.

MS-DOS/UNIX

db_VISTA FEATURES

- Written in C for C.
- Fast B*-tree indexing method.
- Maximum data efficiency using the network database model.
- Multiple key records—any or all data fields may be keys.
- Multi-user capability.
- Transaction processing.
- Interactive database access utility.
- Ability to import and export dBASE II/III and ASCII files.
- 90 day extended application development support.

NO ROYALTIES

SOURCE CODE INCLUDED

db_VISTA PRICE

Single user without source	\$195
Single user with source	\$495
Multi-user without source	\$495
Multi-user with source	\$990

MC/VISA/COD

30 DAY MONEY BACK GUARANTEE

Available for the Lattice, Microsoft, Computer Innovations, DeSmet, Mark Williams, and Aztec C compilers under MS-DOS, and most UNIX systems.

DISCOUNTS ON ALL
LATTICE PRODUCTS

RAIMA

CORPORATION

11717 Rainier Avenue South
Seattle, WA 98178, USA
(206) 772-1515 Telex 9103330300

CALL TOLL-FREE
1-800-843-3313

At the tone, touch 700-992.

Circle No. 266 on Inquiry Card



C STANDARDS

User programs that conform to the Kernighan and Ritchie description of C should continue to compile and execute as before.

dent code. ANSI C will be available for writing applications under MS-DOS, UNIX, or any other system. Such applications may take advantage of any and all system-dependent features in their environment.

On the other hand, the standard should: *Provide a Fighting Chance for Portability*. The standard presents an itemization of implementation-dependent, unspecified, and undefined constructs. Well-formed programs that avoid these constructs will be portable to any ANSI C environment, under any operating system.

In particular, the committee has specified a library that will perform identically on any host system. It is a proper subset of the UNIX library standardized by /usr/group and more recently by the IEEE committee P1003.

Most of these principles can be

Most of the discussion is judged against the shared "spirit of C".

summed up in one phrase: *Preserve the Spirit of C*. Most X3J11 members feel that C does not need any "fixing"—that it is successful and popular precisely because it is powerful, simple, and elegant. At the meetings of the group, most of the discussion is judged against the shared "spirit of C" rather than the interests of specific vendors. As a result, no "factions" have developed so far among X3J11. Argument has sometimes been heated, but the "sides" have proven to be fluid from one question to the next, and issues have been resolved by creative solution more often than by political compromise.

VALUE TO IMPLEMENTERS

The (draft) standard will provide several benefits to implementers of C compilers and interpreters. One is a *clarity* achieved through the resolution of gray areas. For example, the timing of side-effects is specified by certain "sequence points".

The draft also gives implementers greater *opportunities for efficiency*. Until now, the generation of optimized code has been hampered by an uncertainty regarding *method vs. result*. For example, if **c1**, **c2**, and **c3** are **char** variables, the expression **c1 = c2 + c3** involves what K&R called "usual arithmetic conversions"; this widens the **char** values to **int** before the addition. The draft describes the *result* semantics in terms of a hypothetical abstract machine, one which laboriously executes every operation exactly as written. A real implementation is then free to achieve the specified result with a *method* that is more efficient. In the example above, the "usual arithmetic conversions" would be taken to specify the result rather than the method; an optimizing compiler thus might avoid widen-

NEW RELEASE

UNIPRESS EMACS™

EDITOR FOR: UNIX™ /
VMS™ / MS-DOS™

Another in a series of productivity notes on software from UniPress.

Subject: Multi-window, full screen editor.

Multi-window, full screen editor provides extraordinary text editing. Several files can be edited simultaneously, giving far greater programming productivity than vi. The built-in MLISP™ programming language provides great extensibility to the editor.

New Features:

- EMACS is now smaller and faster.
- Sun windows with fonts and mouse control are now provided.
- Extensive on-line help for all commands.
- Overstrike mode option to complement insert mode.
- New arithmetic functions and user definable variables.
- New manual set, both tutorial and MLISP guide.
- Better terminal support, including the option of not using unneeded terminal drivers.
- EMACS automatically uses terminal's function and arrow keys from termcap and now handles terminals which use xon/xoff control.
- More emulation-TOPS20 for compatibility with other EMACS versions, EDT and simple Wordstar™ emulation.

Features:

- Multi-window, full screen editor for a wide range of UNIX, VMS and MS-DOS machines.
- "Shell windows" are supported, allowing command execution at anytime during an edit session.
- MLISP programming language offers extensibility for making custom editor commands! Keyboard and named macros, too.

- "Key bindings" give full freedom for defining keys.
- Programming aids for C, Pascal and MLISP: EMACS checks for balanced parenthesis and braces, automatically indents and reformats code as needed. C mode produces template of control flow, in three different C styles.
- Available for the VAX™ (UNIX and VMS), a wide range of 68000 machines, AT&T family, Pyramid™, Gould™, IBM-PC™, Rainbow™ 100+ and many more.

Price:

	Binary	Source
VAX/UNIX		\$995
VAX/VMS	\$2500	7000
68000/UNIX	395	995
MS-DOS	325	995

For our **Free Catalogue** and more information on these and other UNIX software products, call or write:

UniPress Software, Inc.,
2025 Lincoln Hwy.,
Edison, NJ 08817.

Telephone: (201) 985-8000.

Order Desk: (800) 222-0550 (Outside NJ). Telex: 709418.

European Distributor:

Modulator SA, Switzerland
Telephone: 41 31 59 22 22,
Telex: 911859.

OEM terms available.
Mastercard/Visa accepted.



ing the **char** values if it would mean producing more efficient code.

Another arena for greater optimization is *single-precision arithmetic* (such as **float** arithmetic). The draft allows imple-

menters to make more use of single-precision, thus allowing C to show better performance in an important class of engineering applications.

Implementers have had difficulty introducing optimizations

such as "common subexpression elimination" because C is often used for device drivers and control applications, where access to memory-mapped locations should not be "optimized away". The draft provides a **volatile** type modifier for variables that must be accessed exactly as written. Everything that is not **volatile** is then fair game for all-out optimization.

The draft allows an implementer to provide a "safe" macro (one that evaluates each parameter once) for each library function, in addition to an executable object-code version in the library. This can yield significantly faster execution times in some cases. In fact, it allows any library function to be treated by the compiler as a built-in function for the direct generation of optimal assembler code.

A standard for C will provide a *fixed target for implementations*. Vendors will be able to allocate resources more confidently to the development of a C compiler once they know just what that compiler is supposed to do.

VALUE TO PROGRAMMERS

The opportunity for highly *portable code* is an important benefit of the new standard. Existing compilers have had slightly different features, slightly different implementations, and slightly different libraries.

Program reliability will benefit from *stricter checking*. The standard provides a means for declaring the types of function parameters, allowing the compiler itself to check the agreement of arguments and parameters. (Previously, **lint** was needed for this job.) Another type of checking is provided by the new **const** keyword, which indicates that a variable is read-only and should not be modified. (This allows a compiler to

Basmark BASIC

The first IBM®-PC Compatible BASIC Compiler for UNIX®

Available now:

- VAX®
- Intel® 80286, 8086
- Motorola® 68000

– OEM/Distributor inquiries invited
Binary/Source licenses available



Basmark Corporation

1717 East Ninth • Cleveland, Ohio 44114

Motorola is a registered trademark of Motorola Incorporated, VAX is a registered trademark of Digital Equipment Corporation, IBM is a registered trademark of International Business Machines Corporation, UNIX is a registered trademark of AT&T Bell Laboratories, Incorporated, Intel is a registered trademark of Intel Corporation.

target **const** data for ROM or write-protected memory.)

Important as these advantages are, though, programmers most of all will appreciate the opportunities for *greater efficiency* that the standard promises.

FURTHER INFORMATION

For a point-by-point summary of the differences between K&R Appendix A and the ANSI draft standard, see the 1985 issues of *The C Journal*, PO Box 849, Denville, NJ 07834; 201/989-0570.

The textbook *Reliable Data Structures in C* (Plum Hall, 1985) discusses a programming style that works with current compilers and anticipates the enhancements of ANSI C.

The opportunity for highly portable code is an important benefit of the new standard.

Further discussion of the draft standard has appeared in */c: The Journal for C Users*, Que Corporation, 7999 Knue Rd., Indianapolis, IN 46250.

To join X3J11 or to request information on the status of ANSI C, contact Thomas Plum, Vice-Chair X3J11, Plum Hall Inc.,

1 Spruce Avenue, Cardiff, NJ 08232; 609/927-3770.

The 1985 C Information Bulletin can be obtained by sending a \$20 check payable to "X3 Secretariat" and a self-addressed mailing label to:

C Language Bulletin
X3 Secretariat/CBEMA
311 First St NW, Suite 500
Washington DC 20001

Thomas Plum is Chairman of Plum Hall Inc., and author of several books on the C language. As Vice-Chair of ANSI committee X3J11, Dr. Plum is designated by the committee to handle communications with the public. This article, however, is not an official publication of ANSI X3J11. ■

Another in a series of productivity notes on UNIX™ software from UniPress.

Subject: C Cross Compiler for the 8086 Family.

The Lattice C Cross Compiler allows the user to write code on a VAX™ (UNIX or VMS™) or MC68000™ machine for the 8086 family. Lattice C is a timesaving tool that allows a more powerful computer to produce object code for the IBM-PC™. The compiler is regarded as the finest C compiler for the 8086 family and produces the fastest and tightest code.

Features:

- For your UNIX or VMS Computer.
- Use your VAX or other UNIX machine to create standard Intel object code for the 8086 (IBM-PC).
- Highly regarded compiler produces fastest and tightest code for the 8086 family.
- Full C language and standard library, compatible with UNIX.
- Small, medium, compact and large address models available.
- Includes compiler, linker, librarian and disassembler.
- 8087™ floating point support.
- MS-DOS™ 2.0 libraries.
- Send and Receive communication package optionally available. Price \$500.
- Optional SSI Intel Style Tools. Package includes linker, locator and assembler and creates executables for debugging on the Intel workstation or for standalone environments. Price \$8,550.

Price:

VAX (UNIX or VMS) \$5000
MC68000 3000

For more information on these and other UNIX software products, call or write: UniPress Software, Inc., 2025 Lincoln Hwy., Edison, NJ 08817. Telephone: (201) 985-8000. Order Desk: (800) 222-0550 (Outside N.J.). Telex: 709418. Japanese Distributor: Softec 0480 (85) 6565. European Distributor: Modulator SA (031) 59 22 22.

OEM terms available.
Mastercard/Visa accepted.

CROSS COMPILER FOR THE 8086™ FAMILY

LATTICE® C CROSS COMPILER

INDUSTRY INSIDER

The wave of the future

by Mark G. Sobell

Computers are getting faster, smaller, and cheaper. There is, however, a physical limit to the speed and size that computers can attain. As chips get denser, the thickness of the layers that make them up are measured in atoms and the limiting factor becomes the speed of light. Chip technology is rapidly approaching that limit.

True to form, though, people want machines that run faster. What to do? One increasingly popular approach is to break down the problems that computers solve into tasks that can be executed in parallel (simultaneously), and spread the work over several processors. Any one of the processors can only work so fast, but by combining their efforts, the task at hand can be completed in less time as measured by the clock on the wall.

There is much to suggest that parallel processing will be the wave of the future. But no two manufacturers seem to have the same idea of what parallel processing is.

In this month's column, I explain my understanding of parallel processing by defining many of the terms that describe and distinguish parallel processors. Next month, I will discuss the characteristics that separate several parallel processor machines already on the market.



RELATING PARALLEL PROCESSING TO UNIX

A *multiprocessor* simply is a computer containing more than one processor (CPU). This should not be confused with a *multicomputer*. The difference is that a multiprocessor machine has global memory that all its processors can access while a multicomputer has local memory for each processor. Because the processors in a multiprocessor environment share memory, their functions, specifically their access to memory, must be closely controlled. Processors working within a multicomputer do not require this close supervision.

UNIX, meanwhile, is a *multi-programming* operating system because it can run several *unrelated* programs *concurrently*. On a conventional single-processor

implementation of UNIX, "concurrent" does not mean "simultaneous"; it only signifies that the CPU *appears* to be executing programs simultaneously from the perspective of users and programs. In truth, though, the CPU actually works on only one program at a time.

People also call UNIX a *multi-tasking* operating system. That's because the system allows single jobs to be composed of several tasks working in conjunction with one another. A pipe is one method by which tasks can be joined within a job. Under System V, shared memory and other types of interprocess communication are also possible.

Making use of these terms, you can say that *parallel processing* is "multitasking in a multiprocessor environment". When UNIX is implemented on a *multiprocessor*, "concurrent" suddenly can mean "simultaneous".

As an example, assume you have a job that runs tasks A, B, and C. The tasks are connected by pipes so the output from A goes to B and B's output goes to C. When you run the job on a single-processor computer, all three tasks will appear to be running simultaneously, but the processor can only work on one of these tasks at a time: thus you have multitasking in a single-processor environment. When you run

the job on a multiprocessor computer, all three tasks can actually run at the same time, each on a different processor. This is nothing other than multitasking in a multiprocessor environment: parallel processing. With all else being equal, the computer should spit out an answer almost three times as fast.

Because of its multitasking capabilities, UNIX is a natural vehicle for parallel processing. Many applications that run under UNIX are already structured so that they can exploit a multiprocessor environment.

INTERPROCESS COMMUNICATION AND SYNCHRONIZATION

When several tasks (processes)

There is much to suggest that parallel processing will be the wave of the future.

are working toward a common goal, they usually need a way to share information. When the tasks make use of global memory, the simplest and most efficient way for them to communicate is through shared memory. Other mechanisms, such as signals, can be used where there is no global memory or where it is

inappropriate to use shared memory.

A *semaphore* is a data structure that resides in shared memory and coordinates the actions of several tasks. The simplest semaphore is a *lock*, which is used to ensure exclusive access to a shared data structure. The need for a lock is illustrated by the classic example of two people trying to update an inventory database at the same time. Without a lock, the two users find when they access the database simultaneously that there is only one of a particular part left in stock. Both users then could conceivably sell the same part without being alerted to the actions of the other. That won't be good for either their customers or their

Another in a series of productivity notes on UNIX™ software from UniPress.

Subject: Powerful spreadsheet with NEW ADDED FEATURES.

Q-Calc is an extraordinary spreadsheet for UNIX including extensive math and logic facilities, comprehensive command set, optional graphics, many new ease-of-use features, and the ability to run UNIX programs on spreadsheet data.

Features:

- Fast spreadsheet with large model size, allowing sorting and searching.
- Interfaces with UNIX and user programs via pipes, filters and sub-processes. Data can be processed interactively by UNIX.
- Q-Calc profile mechanism allows the user to store default information, as well as support for terminal-specific profiles. Uses termcap.
- Graphics for bar and pie charts. Several device drivers supported.
- New Features of Version 3.2 include more powerful printing, simpler data input, keybinding definitions, new string operator, bind-to-key, and more.
- Available for the VAX™, Sun™, Masscomp™, AT&T 3B & 7300 Series, Pyramid™, Plexus™, Gould™, Cadmus™, Integrated Solutions™, Cyb™, IRIS™, Callan™, and many more.

Price:

VAX, Pyramid, AT&T 3B/20	Binary \$2500
MC68000™	(with graphics) 3500
	750
	(with graphics) 995
Source Code Available.	

For our **Free Catalogue** and more information on these and other UNIX software products, call or write: UniPress Software, Inc., 2025 Lincoln Hwy., Edison, NJ 08817. Telephone: (201) 985-8000. **Order Desk: (800) 222-0550 (Outside NJ).** Telex: 709418. European Distributor: Modulator SA, Switzerland Telephone: 41 31 59 22 22, Telex: 911859.

OEM terms available. Mastercard/Visa accepted.

SPREADSHEET

Q-CALC

database (which will not be accurately updated). (These users incidentally would need a lock even if they were working in a single-processor environment.)

With a lock, the first user reporting a sale would get the part while the other would get a "Sorry, Charlie" message. That's because when a user updates a database, the software attempts to lock the data structure (in this case, the record pertaining to the hotly pursued part). If the lock is successful, the data structure will be updated and unlocked. If the lock is not successful, it means that the data structure has already been locked by another process. The second process will have to wait until the structure is

unlocked before it can put on a lock of its own. That way, even if there are no parts left by the time the second user puts a lock on the record, the user at least will be advised.

WHY PARALLEL PROCESSING?

All right, so parallel processing can speed things up, but is it worth the added programming and hardware complexity required to support it? Aside from speed, what advantages do manufacturers attribute to parallel processing? One often cited is a better price/performance ratio in certain applications. When you price individual processors, fast ones are much more expensive than slow ones. Multiprocessors allow you to satisfy your requirements for computational power by combining less expensive, slower processors.

A second reason is fault tolerance. Where single-processor machines can be set up to switch to a backup disk drive or printer in the event of a component failure, multiprocessors also can switch to a different processor. The result is a system that is potentially tolerant of CPU failure.

A third point on which parallel processing manufacturers expound is the capacity of their machines to grow in modular fashion. With a system that supports multiple processors, you can start small and increase the capacity of your system as requirements dictate—adding CPUs in just the way you add disk drives to a single-processor system.

So why hasn't anyone come up with a parallel processor before now? Peter Patton in the June 1985 issue of *IEEE Computer* answered the question this way: "While the world around us works in parallel, our perception of it has been filtered through 300

years of sequential mathematics, 50 years of the theory of algorithms, and 28 years of Fortran programming."

The question that comes from this discussion is: now that we are building multiprocessors, what kinds of tasks are best suited to parallel processing? Basically, any set of independent processes is a good candidate for parallel processing. More research is required to determine if jobs or parts of jobs can be broken down into computational chunks that also can be processed in parallel. Recent studies have suggested that the following operations would benefit most from this sort of single-job parallel processing:

- matrix operations.
- image processing and generation.
- signal processing.
- sorting and searching.

Next month, I'll look at how some manufacturers propose to service these sorts of parallel processing needs.

If you have an item appropriate for this column, you can contact Mr. Sobell at 333 Cobalt Way, Suite 106, Sunnyvale, CA 94086.

*Mark G. Sobell is the author of the bestselling book, "A Practical Guide to the UNIX System" (Benjamin/Cummings, 1984) and the new "A Practical Guide to UNIX System V" (Benjamin/Cummings, 1985). He has been working with UNIX for over five years and specializes in documentation consulting and **troff** typesetting. Mr. Sobell also writes, lectures, and offers classes in *Advanced Shell Programming* and **awk**. ■*

Mr. Sobell wishes to thank Sequent Computer Systems, Inc., for the assistance it provided in the development of this article.

UNIX*
JOBS
REGISTRY

National registry of candidates and jobs in the Unix field. Please give us a call; send a resume; or request a free Resume Workbook & Career Planner. We are a professional employment firm managed by graduate engineers.

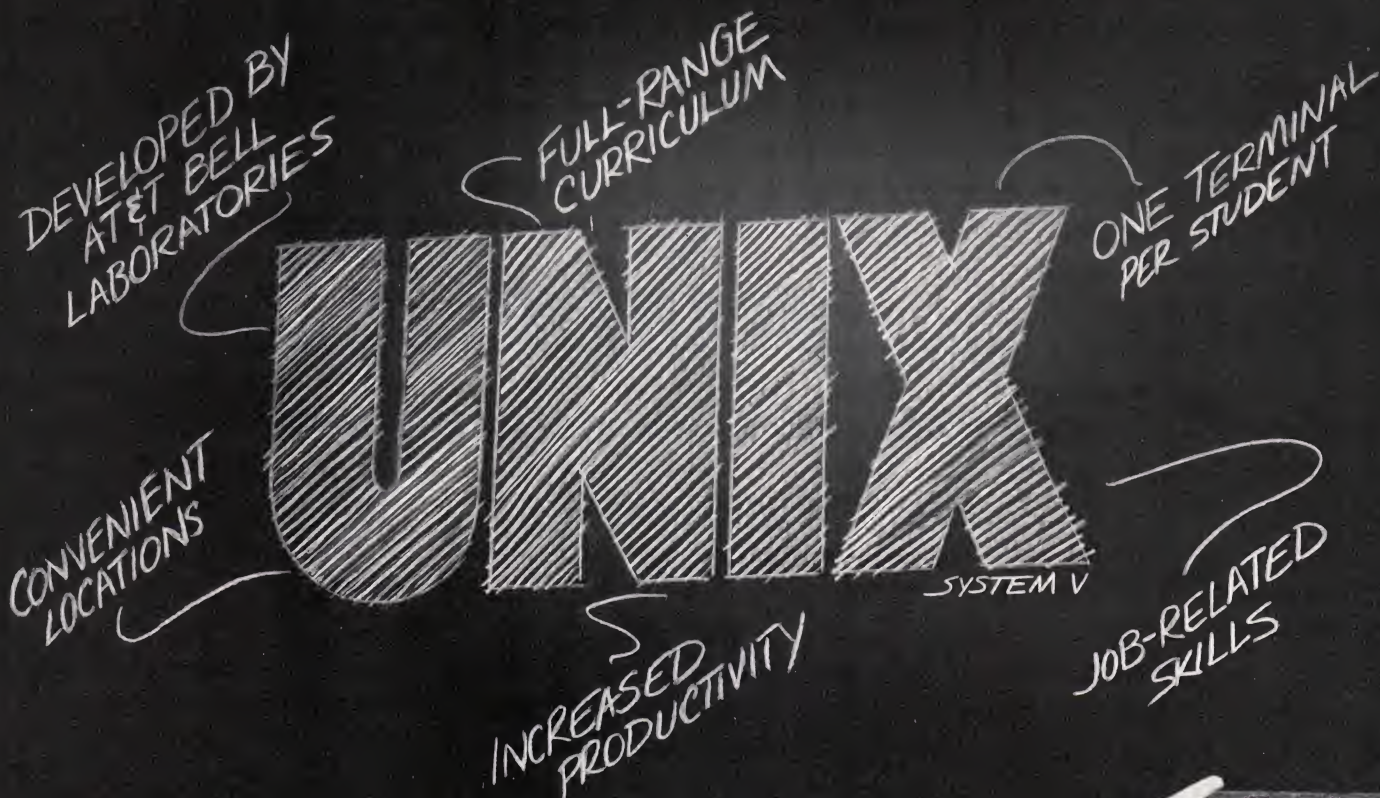
800-231-5920
P. O. Box 19949, Dept. UR
Houston, TX 77224
713-496-6100



Scientific Placement, Inc.

*Unix is a trademark of Bell Labs

Circle No. 270 on Inquiry Card



HANDS-ON TRAINING THAT ISN'T SECONDHAND

When you learn the UNIX™ System directly from AT&T, you learn it from the people who develop it. So all the information you get is firsthand.

For over fifteen years, we've been teaching our people to use the UNIX System—which makes us the best trained to help you learn.

The best training starts at your own terminal. That's why, at AT&T each student gets the use of an individual terminal for real hands-on training.

Take your pick of courses from our extensive curriculum. Whatever your level of expertise, from first-time user to system developer, we have a course that will suit your individual needs. And all our courses are designed to teach you the specific skills that will soon

have you using the UNIX System to organize and expand your computing system for maximum efficiency.

You also get experienced instructors, evening access to training facilities, and your choice of training centers. We can even bring our courses to your company and hold the training at your convenience.

And because we are continually expanding our courses to incorporate the developments of UNIX System V, you're assured of always getting the most up-to-date information.

So take your training from AT&T. And discover the power of UNIX System V—right from the source. **Call us today to reserve your seat or for a free catalog.**
1-800-221-1647, Ext. 355.

Yes, I'd like some firsthand information on all UNIX System training courses.

Name _____

Title _____

Company _____

Address _____

City _____ State _____ Zip _____

Call 1-800-221-1647, Ext. 355
or send coupon to:

AT&T Information Systems
P.O. Box 45038, Jacksonville, FL 32232-9974



AT&T

The right choice.

C ADVISOR

Calling Fortran and Pascal from C (and vice-versa)

by Bill Tuthill

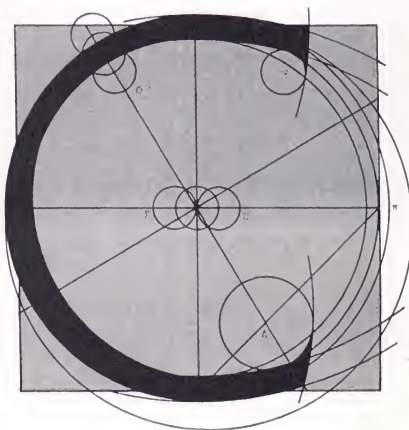
Europeans and academics devise new programming languages all the time—academics because it gives them a convenient research topic, Europeans because they seem to enjoy living beneath a Tower of Babel. But only a few languages really catch on: Fortran for scientists, COBOL for business programmers, BASIC for hobbyists, Lisp for AI researchers, Pascal for students, and C for system programmers. This is not to say these languages are better than new ones coming along. It's just that most new languages aren't so much better that they're worth learning.

Berkeley UNIX provides all the popular languages listed above, except COBOL and BASIC. In addition, it includes a rather poor implementation of APL. The C, Fortran, and Pascal compilers all use the same backend—they share code generator, assembler, and loader—so routines from the three languages can be combined. This column discusses methods for doing so. Note that System V has no Pascal, so remarks made here about Pascal do not pertain to AT&T UNIX systems.

C AND FORTRAN

The **f77** compiler was written at Bell Labs by Stu Feldman (and Peter Weinberger), and distributed for the first time on Version 7. Dave Wasley of UC Berkeley improved the Fortran interface to UNIX, and various other people fixed bugs and worked on speed improvements. The contributions of Don Seeley, now at the University of Utah, are most notable. System V includes much of the work done at Berkeley and elsewhere.

One reason Fortran continues to be so widely



used is that it provides mathematical and statistical libraries, such as IMSL, Linpack, Eispack, and Harwell. C programmers may have occasion to call routines in these libraries. Conversely, Fortran programmers working in a UNIX environment may want to call C library routines and UNIX system calls from inside Fortran programs.

In Fortran, parameters are passed by reference (or address), rather than by value, as in C. Thus, you need to put an ampersand (&) in front of most parameters when calling Fortran routines from C. The **f77** compiler appends an underscore to the names of common blocks, functions, and subroutines in order to distinguish them from C routines or external variables of the same name. This is done because of the different parameter passing conventions. Data types in the two languages correspond in the ways outlined in Figure 1. A Fortran function returning type **integer**, **logical**, or **double precision** returns the same type as the corresponding C function. Note that C functions cannot return a **float** because it would be promoted to **double** first. A **complex** or **double complex** function is equivalent to a C function with a first argument pointing to the address of the return value. So:

```
complex function f(...)
```

in Fortran is equivalent to:

```
f_(temp, ...)  
struct {float r, i;} *temp;
```

in C. Furthermore, a Fortran function returning a

Fortran	C
integer*2 x	short x;
integer x	long x;
logical x	long x;
real x	float x;
double precision x	double x;
complex x	struct {float r, i;} x;
double complex x	struct {double r, i;} x;
character*16 x	char x[16];

Figure 1 — *The ways in which Fortran and C data types correspond.*

character variable is equivalent to a C function with two initial arguments giving the data address and length. Thus:

```
character*14 function g(...)
```

in Fortran is equivalent to:

```
g(result, length, ...)
char result[];
long length;
```

in C, and could be invoked in C as follows:

```
char chars[15];
g(chars, 15L, ...);
```

```
#include <stdio.h>

long unixcmd_(cmd, cmdlen)      /* execute UNIX command */
char *cmd;
long cmdlen;
{
    char buf[BUFSIZ];

    if (cmdlen >= BUFSIZ)
        return(-2L);
    strncpy(buf, cmd, cmdlen);
    /*
     * Fortran strings blank-padded so insert NULL
     */
    buf[cmdlen] = NULL;
    if (system(buf) == 127) /* couldn't exec shell */
        return(-1L);
    return(0L);
}
```

Figure 2 — *An example of how to execute a UNIX shell command from inside a Fortran program.*

As stated above, all Fortran arguments are passed by reference. Also, **character** parameters require a trailing argument giving the length of the string. Thus, the Fortran call:

```
external func
character*8 str
integer num(3)

call sam(func, num(2), str)
```

is equivalent to the C call:

```
int func_();
char str[8];
long num[3];

sam_(func_ , &num[1], str, 8L)
```

The string is eight characters long. Note that C arrays are indexed beginning at zero, whereas Fortran arrays begin at one, so we must pass **num(2)** to the Fortran subroutine, but **num[1]** to the C function. Two-dimensional arrays can be a stumbling block, because C arrays have row-major ordering, while Fortran arrays have column-major ordering. In other words, Fortran stores array elements with the first subscript varying most rapidly, while C stores them with the final subscript varying most rapidly.

The Fortran I/O library is implemented on top of C's standard I/O library. Every open unit in a Fortran program has an associated FILE structure. The **stdin**, **stdout**, and **stderr** streams are easy to share because they don't require explicit references, but other streams (units) opened from Fortran are difficult to share, although it can be done by writing a C routine such as **getfd()** to ascertain the file descriptor.

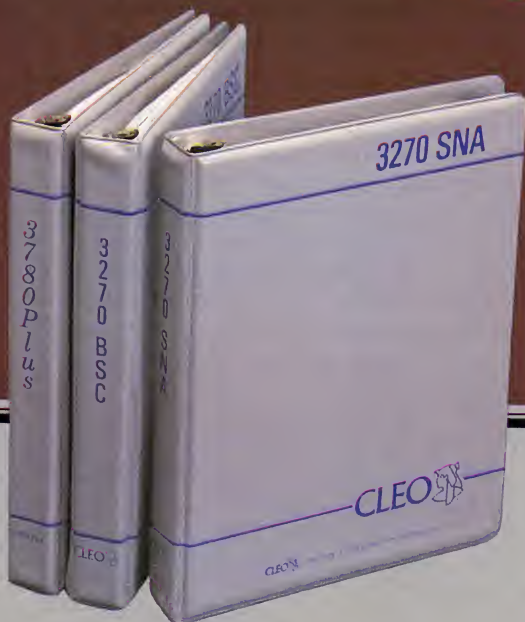
Let's suppose we want to execute a UNIX shell command from inside a Fortran program. There's already a library routine, **system(3f)**, for doing this. But for the sake of an example, let's change the name and code the routine in C as shown in Figure 2. Note the underscore appended to the function name. To call this routine from Fortran, all we need is something like:

```
call unixcmd('ls -l')
```

We place the **unixcmd_()** routine in a file named **unixcmd_.c**, and compile the Fortran program and the C code as follows:

```
% f77 prog.f unixcmd_ .c
% a.out
```


CLEO is your SNA or BSC Gateway



Connect your IBM, Apple, Tandy, Zenith, A.T.&T., Hewlett-Packard, Televideo, NCR, IMS, SUN, or other DOS or UNIX-based system to another micro or to your mainframe with CLEO Software.

Now you can connect your PC LAN, too!

For details call: 1(800) 233-CLEO
In Illinois 1(815) 397-8110

CLEO

CLEO Software
a division of Phone 1, Inc.
1639 North Alpine Road
Rockford, IL 61107
TELEX 703639

Circle No. 298 on Inquiry Card

CLEO and 3780Plus are registered trademarks of CLEO Software.
IBM is a registered trademark of International Business Machines Corporation; Apple is a registered trademark of Apple Computer; UNIX is a registered trademark of A.T.&T. Technologies, Inc.

blanks; UCSD Pascal uses counted byte arrays instead. To make string manipulation easier, some Pascal compilers (such as Sun's **pc**) have implemented the ISO standard for conformant arrays, with the extension that literal strings will be null-terminated when passed as conformant array value parameters. This makes the program easier to code, and more efficient, as demonstrated in Figure 5.

Unlike the Fortran compiler, the Pascal compiler doesn't know how to deal with C programs. In our example, this is no problem, because we don't have to write a wrapper routine in C. But in programs with complex data formats, wrapper routines are often required. To compile and load such programs, you have to invoke the C compiler, then the Pascal compiler. Here's how we would compile and run the Pascal program above:

```
% pc list.p
% a.out
```

Pascal automatically includes all the standard C library routines. We execute the program simply by typing *a.out*.

Suppose, on the other hand, we wanted to call a Pascal library routine from a C program. There aren't nearly as many good Pascal libraries as Fortran libraries, but it's possible that some companies have developed Pascal libraries that will someday need to be linked with C programs. Let's use the same example as with Fortran: a function to compute the area of a circle, given the radius. Remember that Pascal doesn't have an intrinsic power function, but with squares the work-around is easy:

```
function area(r: real): real;
begin
    area := 3.1415926535897932 * (r * r)
end;
```

From a C program, we could call this function as follows:

```
double r, a, area();

r = radius;
a = area(r);
```

Note that we declare everything as **double precision**, because most Pascal compilers don't have a data type for **single precision**. Here's how to compile the Pascal function and the C program:

```
% pc -c area.p
% cc prog.c area.o -lpc -lm
% a.out
```

```
program list(input, output);
    type string = packed array[1..512] of char;
    var s: string;

procedure system(var cmd: string);
external:

begin | main |
    s := 'ls -l';      | Pascal strings are blank padded |
    s[6] := chr(0);   | so we put a NULL there, as in C |
    system(s)
end.
```

Figure 4 — An example of how to execute a UNIX shell command from inside a Pascal program.

```
program list(input, output);

procedure system(cmd: packed array[1b..ub: integer] of char);
external:

begin
    system('ls -l')
end.
```

Figure 5 — An example of how a UNIX shell command can be executed from inside a Pascal program using a Pascal compiler that has implemented the ISO standard for conformant arrays.

We load the Pascal library, in case there are **writeln()** statements or something of the sort, and we also load C's math library. Our example requires neither, but it's best to be on the safe side. As before, we execute the program by typing *a.out*.

CONCLUSION

For ease of implementation and maintenance, both the Fortran and Pascal compilers use the same code generator, assembler, and loader as the C compiler. This has a beneficial side-effect: routines in the three languages are mutually callable. Although not discussed here, Fortran also can be called from Pascal, and vice-versa. This is an example of the UNIX tool philosophy at work—a few small tools that provide great flexibility and functionality.

Bill Tuthill was a leading UNIX and C consultant at UC Berkeley for four years prior to becoming a member of the technical staff at Sun Microsystems. He enjoys a solid reputation in the UNIX community earned as part of the Berkeley team that enhanced Version 7 (4.0, 4.1, and 4.2BSD).

UNIX & CTM

HANDS-ON SEMINARS

A Complete Curriculum for: End Users • Management • Applications Staff • Technical Support

COURSES	LONDON	BOSTON	CHICAGO	DALLAS & SAN FRANCISCO	LOS ANGELES	NEW YORK & SOMERSET	TORONTO & ORLANDO	WASHINGTON, D.C.	TUITION	SEQUENCE TUITION†
UNIX Overview	Oct 8 85 Dec 3 85 Feb 11 86 Apr 15 86 June 17 86	Dec 17 85 Mar 18 86	Mar 11 86	Oct 15 85 Jan 28 86 Apr 29 86	Mar 11 86	Oct 1 85†† Feb 4 86 Apr 15 86	Jan 28 86 Apr 29 86	Oct 1 85 Dec 1 85 Jan 14 86 Mar 11 86 May 20 86	\$225	\$860
UNIX Fundamentals for Non-Programmers*	Oct 9-11 85 Dec 4-6 85 Feb 12-14 86 Apr 16-18 86 June 18-20 86	Dec 18-20 85 Mar 19-21 86	Mar 12-14 86	Oct 16-18 85 Jan 29-31 86 Apr 30-May 2 86	Mar 12-14 86	Oct 2-4 85†† Feb 5-7 86 Apr 16-18 86	Jan 29-31 86 Apr 30-May 2 86	Oct 2-4 85 Dec 18-20 85 Jan 15-17 86 Mar 12-14 86 May 21-23 86	\$735	
UNIX Fundamentals for Programmers*	Oct 14-16 85 Dec 9-11 85 Feb 17-19 86 Apr 21-23 86 June 23-25 86	Mar 31-Apr 2 86	Mar 17-19 86	Oct 28-30 85 Feb 3-5 86 May 5-7 86	Mar 17-19 86	Oct 7-9 85†† Feb 10-12 86 Apr 21-23 86	Feb 3-5 86 May 5-7 86	Oct 7-9 85 Jan 20-22 86 Mar 17-19 86 Jun 2-4 86	\$735	\$1125
Shell as a Command Language*	Oct 17-18 85 Dec 12-13 85 Feb 20-21 86 Apr 24-25 86 Jun 26-27 86	Apr 3-4 86	Mar 20-21 86	Oct 31-Nov 1 85 Feb 6-7 86 May 8-9 86	Mar 20-21 86	Oct 10-11 85†† Feb 13-14 86 Apr 24-25 86	Feb 6-7 86 May 8-9 86	Oct 10-11 85 Jan 23-24 86 Mar 20-21 86 June 5-6 86	\$490	
'C' Language Programming*	Oct 21-25 85 Dec 16-20 85 Feb 24-28 86 Apr 28-May 2 86 Jun 30-July 4 86	Apr 7-11 86	Sept 9-13 85 Mar 31-Apr 4 86	Nov 11-15 85 Feb 10-14 86 Apr 12-16 86	Sep 9-13 85 Mar 31-Apr 4 86 Aug 4-8 86	Oct 21-25 85†† Feb 17-21 86 Apr 28-May 2 86	Feb 10-14 86 Apr 12-16 86	Oct 21-25 85 Jan 27-31 86 Mar 31-Apr 4 86 Jun 9-13 86	\$1225	
Shell Programming*	Sep 2-3 85 Oct 28-29 85 Jan 6-7 86 Mar 3-4 86 May 12-13 86	Apr 14-15 86	Sep 16-17 85 Apr 7-8 86	Nov 18-19 85†† Feb 17-18 86 May 19-20 86	Sep 16-17 85 Apr 7-8 86	Oct 28-29 85†† Feb 24-25 86 May 5-6 86	Feb 17-18 86 May 19-20 86	Oct 28-29 85 Feb 3-4 86 Apr 31-May 1 86 Jun 16-17 86	\$490	\$1125
Using Advanced UNIX Commands*	Sep 4-6 85 Oct 30-Nov 1 85 Jan 8-10 86 Mar 5-7 86 May 14-16 86	Apr 16-18 86	Sep 18-20 85 Apr 9-11 86	Nov 20-22 85 Feb 19-21 86 May 21-23 86	Sep 18-20 85 Apr 9-11 86	Oct 31-Nov 1 85†† Feb 26-28 86 May 7-9 86	Feb 19-21 86 May 21-23 86	Oct 30-Nov 1 85 Feb 5-7 86 Apr 2-4 86 Jun 18-20 86	\$735	
UNIX Internals	Sep 9-13 85 Nov 4-8 85 Jan 13-17 86 Mar 10-14 86 May 19-23 86	Apr 21-25 86	Sep 23-27 85 Apr 14-18 86	Dec 2-6 85 Feb 24-28 86 Jun 2-6 86	Sep 23-27 85 Apr 14-18 86	Nov 11-15 85†† Mar 3-7 86 May 12-16 86	Feb 24-28 86 Jun 2-6 86	Nov 11-15 85 Feb 10-14 86 Apr 14-18 86 Jun 23-27 86	\$1375	
UNIX Administration*	Sep 18-20 85 Nov 11-13 85 Jan 22-24 86 Mar 19-21 86 May 28-30 86	Sep 10-12 85 Apr 29-May 1 86	Oct 1-3 85 Apr 22-24 86	Dec 10-12 85 Mar 4-6 86 Jun 9-11 86	Oct 1-3 85 Apr 22-24 86	Nov 19-21 85†† Mar 11-13 86 May 20-22 86	Mar 4-6 86 Jun 9-11 86	Nov 19-21 85 Feb 18-20 86 Apr 22-24 86	\$735	
Advanced 'C' Programming Workshop*	Sep 23-24 85 Nov 18-19 85 Jan 27-28 86 Mar 24-25 86 Jun 2-3 86	Sep 16-17 85 May 5-6 86	Oct 7-8 85 Apr 28-29 86	Dec 16-17 85 Mar 10-11 86 Jun 16-17 86	Oct 7-8 85 Apr 28-29 86	Dec 2-3 85 Mar 17-18 86 Jun 2-3 86	Mar 10-11 86 Jun 16-17 86	Dec 2-3 85 Feb 24-25 86 Apr 28-29 86	\$490	\$1125
Advanced 'C' Programming Under UNIX*	Sep 25-27 85 Nov 20-22 85 Jan 29-31 86 Apr 2-4 86 Jun 4-6 86	Sep 18-20 85 May 7-9 86	Oct 9-11 85 Apr 30-May 2 86	Dec 18-20 85 Mar 12-13 86 Jun 18-20 86	Oct 9-11 85 Apr 30-May 2 86	Dec 4-6 85†† Mar 19-21 86 Jun 4-6 86	Mar 12-14 86 Jun 18-20 86	Dec 4-6 85 Feb 26-28 86 Apr 30-May 2 86	\$735	
Berkeley Fundamentals and 'csh' Shell*	Sep 30-Oct 4 85 Nov 25-29 85 Feb 3-7 86 Apr 7-11 86 Jun 9-13 86	Sep 23-27 85 May 12-16 86	Oct 21-25 85 May 5-9 86	Jun 23-27 86	Oct 21-25 85 Feb 3-7 86 May 5-9 86	Dec 9-13 85†† Mar 31-Apr 4 86 Jun 9-13 86	Jun 23-27 86	Dec 9-13 85 Mar 3-7 86 May 5-9 86	\$1225	

*Including hands-on training workshops ††UNIX is a trademark of Bell Laboratories †Savings for consecutive seminar dates

CALL FOR DETAILS ON: ON-SITE SEMINARS • VIDEO-BASED TRAINING • INTERACTIVE VIDEODISC TRAINING
To reserve your seminar space now or for additional information, call: (800) 323-UNIX or (312) 987-4084

Three factors make the Computer Technology Group the experts in UNIX and 'C' language training:

- Experience, through training thousands of students worldwide in live seminars, with thousands more using our video training at their locations.
- Extensive Curricula Supporting All UNIX Versions, creating a client base of manufacturers, software developers and end users.
- Quality of Instruction, with instructors and course developers who are experts in teaching UNIX and 'C', as well as in designing and implementing a variety of UNIX-based systems.

COMPUTER TECHNOLOGY GROUP

Telemedia, Inc.
 310 S. Michigan Ave., Chicago, IL 60604
 The Leading Independent UNIX System Training Company

Circle No. 251 on Inquiry Card

RULES OF THE GAME

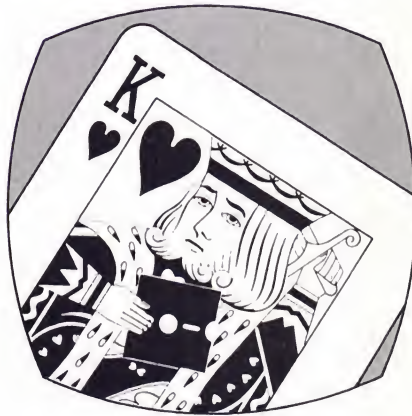
Just one of those things

by Glenn Groenewold

The very rich are indeed different from you and me, as Scott Fitzgerald is supposed to have observed. Long before the rest of us, they were aware that matters of money and property must not be left to chance. Thus marriages among the wealthy invariably have been preceded by *antenuptial* agreements. These legal documents carefully have defined who was to own what so that family fortunes and business interests would not be diluted through inheritance, improvidence, or—unthinkable to the *hoi polloi*—divorce.

Cold-blooded as these written agreements may have appeared to the general populace, they have been considered a necessity to ensure commercial and financial stability, and by and large they have fulfilled that purpose. However, these documents also have served a second function: as blueprints for the orderly dissolution of the marital relationship.

Today marriage is not the only institution in our society that enjoys less permanence than it once did. Increasingly, the employee who works for the same company 30 or 40 years, collects a gold watch, and goes out to pasture is becoming a rarity. Sometimes this is not the employee's choice: in our era, business enterprises often are born, come



to vigorous maturity, and then die or merge within the space of a few years. Still, frequently it's the employee who makes the decision that it's time to move outward and upward.

The implications of this evolution in the job market are significant both for employees and employers. It's in the best interest of each that they realize at the time an employment relationship is created that *it probably isn't going to be permanent*. In short, a contract of employment should be approached as though it were an antenuptial agreement for a marriage that may not last.

The obvious difficulty with this approach is that it often runs counter to human psychology. When we embark on a new job, it's natural to want to feel that we've found our niche. Even the

officers of large companies may have a need to think of the firm's employees as a loyal corporate "family". The gold-watch mindset is very much with us still. But then, why not continue to indulge ourselves in such fantasies so long as they make us comfortable?

Well, one of the things that puts lawyers on approximately the same level of popularity as undertakers is their propensity for pointing out the unpleasant realities of life—such as telling people who don't want to think about their demise that they nevertheless should prepare wills. Another unpleasant reality is that employees and employers need to contemplate the eventual termination of their relationship even as they begin it.

WHY IT MATTERS

Ordinarily, the gold-watch retiree actually *retired*. He or she did not launch a new business in the same field as the former employer, nor did he or she become a consultant for one of its competitors. Thus, if the retiree knew any of the employer's trade secrets, which in the old days wasn't all that likely, it was of little consequence. Today, it can matter a great deal, especially in the computer industry.

Moreover, an evolutionary al-

BREAK THROUGH THE SOFTWARE BOTTLENECK

UNIX™ APPLICATION DEVELOPMENT

TODAY is far more than the awkward collection of tricks and tools that are often labelled "4GL". TODAY provides a **COMPLETE application development environment** that will revolutionize the way you develop and maintain applications. **No UNIX* systems knowledge is necessary.**

Let's put it frankly: developing an application is a costly proposition. You'll need a highly skilled team of designers, analysts and programmers, and several man-years to get things off the ground. And that's not to mention the on-going costs of documentation, customization and maintenance!

TODAY tackles these problems through a new methodology with high performance architecture and a comprehensive range of features. It's so quick and easy to use that TODAY developers can do the whole job—design, analysis, development and documentation.

TODAY provides a comprehensive range of features that keep application building easy while optimizing development resources:

- Powerful recursive logic and Decision Tables
- Synonyms, Menus, Prompts, Helps and Defaults for streamlined definitions
- Screen Painter
- A Report Generator which includes a Painter



- Push-button Self-documentation
- Audit Trails
- Source-code security through run-time only configurations
- Developed Applications instantly portable across UNIX* systems

Because definitions are **Dictionary-based**, any changes are easily made in one central location. A key feature, "**tailoring**" lets you alter an application — perhaps to customize it for a particular site or user — without affecting the original version. If required, applications can be set up as Models (Prototypes) and later enhanced to grow and change with the business. Tailoring versions is the perfect solution for quickly generating multiple applications based on one Model.

TODAY runs under UNIX* or UNIX*-compatible operating systems on **super-mini down to micro business computers** using any of a range of databases. And if that's not enough, TODAY is backed by 14 man-years of research and development and the confidence of users who are breaking time zones in software development.

See us at UNIX Expo, New York City, September 18-20, Booth 1303.

bbj Computer Services, Inc.
2946 Scott Blvd.
Santa Clara, CA 95054
Telephone: (408) 727-4464

Circle No. 284 on Inquiry Card

teration in the nature of the employer-employee relationship itself has been taking place. Originally, our laws permitted an employer to discharge almost any employee at will. Never mind that the employee had served faithfully and was only a year away from getting that gold watch. Gradually this has been changing, nowhere more dramatically than in California, where liberal courts are blazing the trail. Today it's extremely difficult in the Golden State for an employer unilaterally to replace a "permanent" employee with another, except in cases of actual nonperformance or misconduct.

This means that often the preferred way of getting rid of an

unwanted employee is to induce a voluntary departure—the pain of separation being soothed by a generous administration of green salve. The interests of both the employer and employee with respect to such things as protecting the employer's trade secrets and ensuring the employee's ability to engage in competing enterprises can be furthered as part of the termination agreement between the two. Provision for this kind of ultimately amicable parting may be made most effortlessly in the initial hiring agreement.

IN GENERAL


As a newly-hired employee, you should always be sure to do two things for your own pro-

tection. First, read *everything* that's presented for your signature. Frequently this can be a pain, but it's the only way to catch language in the employment documents that you can't live with. Bear in mind that often these provisions can be negotiated. Suppose you plan to develop software at home on your own time with your own equipment, and you're handed the company's "standard" printed employment agreement that says all of your creations throughout the duration of the employment are to be turned over to your employer. Although this part of the agreement probably wouldn't hold up in court under such circumstances, it would be best to attempt to get it modified. If necessary, consult a lawyer to learn what kind of language you can accept. Why risk a nasty lawsuit later on?

Second, *keep* everything pertaining to your employment, including the stuff that strikes you as garbage. This means such items as the orientation packet ("Welcome to the XYZ Family...") because there might be something in there that later on a judge might decide was a part of your employment contract. It's a good idea to put obviously important documents, like the nondisclosure agreement you've signed, in a safe deposit box. Other items, such as orientation materials, should be kept in your files at home—not at the office.

For the employer, the checklist is a lot longer. While it's especially easy for a small employer to overlook important details—what company with only two or three employees has a personnel department?—even large companies are not immune. After all, most sizable enterprises started out small. In the process of having "just grown", all sorts of ad-

CHOOSING
A
UNIX™
VENDOR



BEFORE YOU DO,
ASK THESE QUESTIONS:

1. "Do you have an unparalleled reputation for supporting end-users?"
2. "Have you selected only the **best** Unix hardware and software to sell?"
3. "Have you been offering timeshared Unix applications packages to hundreds of users for more than 3 years?"
4. "Have you been **using** Unix for 10 years?"

IF YOU ASKED BASIS, WE'D SAY YES . . . **FOUR TIMES!**

BASIS

SPECIALISTS IN UNIX COMPUTING

1700 Shattuck Avenue Berkeley, California 94709 415 841 1800

UNIX is a trademark of AT&T Bell Laboratories.

Circle No. 265 on Inquiry Card

ministrative subtleties may have been passed by. (One hears horror stories of long-time employees who've *never* even been required to sign nondisclosure agreements.) And a firm's documents and procedures may have developed piecemeal, with something borrowed from here and something copied from there, to the point where they're actually inconsistent. Finally, some of the more antique of these documents and procedures may now be illegal. The discussions of the following items spotlight some points employers should keep in mind when bringing new employees into the organization.

WRITTEN AGREEMENTS

The recent tendency on the part of the courts has been to elevate verbal employment agreements to the point where they have nearly the clout of written contracts. But oral agreements have a huge disadvantage for both parties: there's no way to be certain what they comprise until some judge renders a decision. For key employees, at least, it's desirable that the parties' understanding of the terms of employment be put in writing. However, it's hazardous to attempt to use the same "boilerplate" contract in all situations: if the agreement is full of language that obviously doesn't apply while ignoring areas that should have been covered, a court might throw it out.

Besides defining the period of employment and providing terms for a separation, a written agreement can specify the sort of activities that are considered inconsistent with the employment, and can specify any limitations on the employee's right to engage in competing enterprises after termination. Such provisions, however, must be carefully written so that they don't run afoul of legal prohibitions.

From the employer's point of view, one advantage of a comprehensive written agreement is that its provisions can be coordinated to accomplish things that could not legally be done piecemeal. For instance, an otherwise unenforceable agreement not to compete might very well hold up if it is coupled with the employer's agreement to retain the former employee as a consultant—with suitable remuneration—for a specified period after termination.

NONDISCLOSURE AGREEMENTS

Not only is it vital that the employer require its employee to sign appropriate agreements to

keep confidential the employer's trade secrets and proprietary information, but the employer also must take special care to comply with any licensing agreements to which it is a party. Typically, licensees are required to obtain nondisclosure agreements from employees who have contact with the licensed product. Failure to comply with this requirement is a breach of the licensing agreement. UNIX licensees should be aware of the provisions of AT&T's licensing agreements in this regard.

ORIENTATION MATERIALS

Even today, the majority of employees probably regard most of the items traditionally included

UNIX POWER

100,000 software developers can't be wrong.*

UNIX is the chosen operating system for more than 100,000 software developers because it has the power they need. But developers aren't the only people who need computing power. Any business that wants multi-users to access the same files at the same time or wants to simultaneously run multi-task operations . . . needs UNIX. At Dynacomp, we offer UniPlus+® System V by UniSoft Corp. For \$1495. U.S. dollars you can run UNIX on the CompuPro® 816/E™ . . . a powerful 68K S-100 bus computer system that

maximizes its memory for multi-user/multi-task operations.

UniPlus+ includes all the standard UNIX System V features PLUS performance enhancements found only in UniPlus+. These features increase the portability, flexibility, and performance of UNIX, allowing an affordable operating system for program development, text preparation, and general office use.

If it's time for you to upgrade to UNIX, call your local Full Service CompuPro System Center in the United States or call

Dynacomp in Canada for complete details.

FROM

210 W. Broadway
Vancouver, B.C.
V5Y 3W2
(604) 872-7737

DYNACOMP

COMPUTER SYSTEMS LTD.

46-6535 Mill Creek Dr.
Mississauga, Ont.
L5N 2M2
(416) 826-8002

*AT&T estimates that there are more than 100,000 people currently developing software under UNIX. Dynacomp serves all of Canada and parts of Asia and the Pacific Rim. Call us for details and information on our full product line including Plexus. • UNIX is a trademark of Bell Laboratories, Inc. CompuPro is a registered trademark and System 816/E is a trademark of Viasyn Corp. UniPlus+ is a registered trademark of UniSoft Corp. AT&T is a registered trademark of AT&T Information Systems.

Circle No. 268 on Inquiry Card

in the typical "welcome aboard" packet as PR junk. Once they've determined where their parking area is and where their medical bills should be sent, the tendency is to discard this "welcome" material, or at least to shove it in the bottom desk drawer where it will never be looked at again.

However, lawyers are becoming increasingly aware of the use to which they can put this seemingly innocuous material. They're especially intrigued by anything in the way of an employee handbook or manual, because these in particular have often been determined by judges to constitute part of the employment agreement.

For instance, does the orientation material seem to promise that particular benefits will be provided on a continuing basis? Does it state that it's company policy to permit employees to work up to a particular age so long as their job performance is satisfactory? Does it make mention of grievance or disciplinary procedures?

Sometimes management is unaware of the representations made by its personnel department in an endeavor to persuade newly-recruited employees that XYZ Company is the greatest place on Earth to work. Given the direction the courts have taken, the safest course is to regard these orientation materials as possible time bombs. *All* such employee handouts should be reviewed by a lawyer before their dissemination.

EMPLOYEE EVALUATIONS

Though they may go on indefinitely, periodic employee appraisals by supervisors, which many large firms require, are really a continuation of the hiring process. The difficulty with these evaluations, as anyone who's ever been on either the receiving or the preparing end is aware, is

The pain of separation can often be soothed by a generous administration of green salve.

that they're usually viewed as an embarrassing nuisance. As a result, evaluations lower than "excellent" are rarely made. This makes them worse than nothing from the employer's standpoint because an employee contesting a dismissal can use these glowing evaluations as evidence in court to show that he or she *was* performing the job in excellent fashion.

It comes down to this: if an employer insists on some sort of formal periodic evaluation of its employees, it should devise a system that will mean something. Exactly what this might be is not clear; some commentators have suggested that only essay-type evaluations be used. Others maintain that even these are not worth the trouble.

DISCIPLINARY PROCEDURES

There's increasing agreement among the legal fraternity that it's essential for an employer to have a consistent, fair procedure for notifying employees that their performance is deemed unsatisfactory, and that failure to shape up will result in discipline or dismissal. Though the procedure normally is not set forth in the written employment agreement itself, it can be incorporated by reference.

Whatever it may be, the disciplinary procedure must be strictly

adhered to, or else the employee will have a basis for contesting the employer's adverse actions in court. Therefore, the procedure should not be unduly burdensome. Yet it can't be so peremptory as to be manifestly unfair to the employee. Clearly, it's wise for an employer to consult carefully with an attorney before deciding on a particular disciplinary scheme.

JURISDICTION

In general, employer-employee controversies are matters for the state courts, although there are significant exceptions, notably claims of discrimination in violation of federal civil rights laws. Not everything that's been said here would apply in every state.

But jurisdiction in employment matters is a tricky thing, given the interdependence of our present-day economy. For instance, under some circumstances California courts would have legal authority to apply its laws to, say, an employee working in Texas. Was the job offered and accepted over the telephone while the employee was attending a conference in California? If so, the contract of hire was made in California, and its courts would have jurisdiction.

Here is just another example of the anomalies that result when a nationwide economy operates within an 18th Century federalist system. This makes it even more important that employers and employees take care to spell out the terms and conditions that govern their employment relationships.

Glenn Groenewold is a California attorney who devotes his time to computer law. He has served as an administrative law judge, has been active in trial and appellate work, and has argued cases before the state Supreme Court. ■



The Quality GKS

PRIOR's GKS has been implemented in the **C** language and is available to **OEM's** and **END USERS** under **UNIX** and UNIX-like systems as well as **VAX/VMS**. GKS/C is supported under **PC-DOS/MS-DOS**.

OEM's receive **SOURCE CODE** (including VDI device support), well-written manuals, superbly commented listings, all available **DRIVERS** and the first year **UPDATE AND MAINTENANCE** for \$12,000* plus royalties. Our current release includes code optimizations, improved execution speed and reduced memory requirements.

* Price subject to change without notice.

GKS/C is supported by PRIOR's **75** in-house Software/Graphics experts. GKS/C is available to level **2C** (UNIX).

3-Dimensional support will be available by first quarter 1986.

For more information, call or write:



PRIOR Data Sciences
39 Highway 7, Ottawa, Ontario K2H 8R2 Canada
613-820-7235 — Telex: 053-3356

Circle No. 227 on Inquiry Card

Distributor inquiries welcome.

UNIX is a registered trademark of AT&T Bell Laboratories. VAX/VMS is a registered trademark of Digital Equipment Corporation. PC-DOS is a registered trademark of International Business Machines. MS-DOS is a registered trademark of Microsoft Corporation.

DEVIL'S ADVOCATE

Keep those cards and letters coming

by Stan Kelly-Bootle

I was delighted to see those "Dear Editor" letters in the *The Last Word* feature of recent UNIX REVIEWS, whereby you, the readers, the very lifeblood without which etc., can participate in our historic mission etc.

Ah, I can see my new WWB (Writers Workbench) is not performing as promised. I was told that the command **etc.** (extra terms concatenator) would automatically generate apposite, in-line strings, known as *ficelles justes* or *contextensions*, until you hit CTRL-S (Suspend), CTRL-Q (Qeep going?), naturally, causes a resumption of **etc.** output. To avoid any confusion, the Berkeley version uses CTRL-H (*Halt*) and CTRL-R (*Resume*).

The **etc.** syntax (not to be confused with */etc* directories) is easier to manage than **sed** syntax. For example:

etc.

with no arguments will amplify your text with a string of up to 64 relevant characters (including punctuation), but you retain the option to pause, edit, continue, or exit. Alternatively:

etc. '/s4/ /:/'

will provide four complete sentences, pausing at each full



stop—unless you intervene manually. All these textual interpolations come, by default, from the standard WWB input file of context-sensitive data. However:

```
etc. '/c3/ /:/' < yourfile
```

allows you to use your own private fund of fillers, clichés, and placebos. In this last example "c3" indicates that three of your own clauses should be selected with what we call the "Reagan option" (pause on semi-colon).

If, flying in the face of the *The Chicago Manual of Style* (which I thought was a baseball book until last season's playoffs), you need a *genuine* unexpanded "etc.". WWB provides a diacritical tactic, but the exact sequence escapes me at the moment.

Those who question the value of all this and claim that it's quicker to type in one's own contextensions are clearly not true UNIX addicts. Besides, for writers in a hurry, especially those paid by the *em* space, it is a joy to produce 10 pages of passable rubbish with a quick: "The computer has changed the way we work, play, etc. '/p10/'".

But back to the correspondence columns I started to speak of earlier. Mail to editors has always been my favorite branch of literature, owing to the paucity of my attention span. A friend of mine who used to edit *Picture Post* (a British version of *Life* magazine) told me once that if things were quiet, staff members would write bogus letters to fill the blanks. The subjects were chosen to invoke torrents of genuine correspondence. His most successful invention, which by a stroke of genius managed to combine pets and religion, purported to be from a widow who was upset that her dog was not allowed in with her when she went to Chapel. For several years thereafter, writers such as *Col. Rtd.*, *Anti-Vivisectionist* and *Mother of Eight* sent in their weekly epistles arguing the proposition that "Animals have Souls".

Fortunately, UNIX REVIEW has no need to stoop to such trickery.

We simply ask Bill Tuthill to use **goto** in a program example...and, whooosh, the mail room is bedlam. "Bill," we say, "page 104 is looking rather bleak. D'you think you could run up something, how can we put it, that's a tad unstructured?" Never fails.

Many have written asking where I stand on the **goto** controversy that simply refuses to go away. At the terribly low level of my current programming efforts, all I can promise is that I will forego the old **goto** construct as soon as Motorola can produce a useful, compatible 68000 with no **JMP**, **BRA**, or **Bcc** instructions. Should Motorola tackle the project, rest assured that the big Coke marketing fiasco will not be forgotten. Forewarned, Motorola will know better than to change the 68000 architecture to match that of the 8086/8088.

Insofar as I understand the **goto** discussion at higher levels, I feel that I am essentially a neo-Knuthian, not untempered by a few hammer-blows from the post-Dijkstrastranist Jacopinities. But can it really be that simple? As I have explained elsewhere (*The Devil's DP Dictionary*, McGraw-Hill, 1981), the Bible makes it clear that the **goto** is an integral part of the inescapable Babel Punishment Package. ("Goto, let us go down and confound their language." —Genesis 11:7.) It seems such a nice idea to be able to transfer control to some as yet unwritten part of your program and then break for coffee. Yet, verily, your sins will be manifest even unto the next generation.

Recent renewed interest in Reduced Instruction Sets reminds me that in the early poor-but-happy EDSAC days, we used to ask, "If you were restricted to just two machine instructions, which would you choose?" The cunning answer was, and still is (I believe), **SUBTRACT** and **BRANCH-NEGA-**

All I can promise is that
I will forego the old
goto construct as
soon as Motorola can
produce a useful,
compatible 68000
with no **JMP**, **BRA**, or
Bcc instructions.

TIVE. There you are! Stuck with the dreaded **goto**, unless you avoid multiplication and division. And for all you Budding Zens out there, what if you were reduced to a single instruction? Would it be the elusive *comefrom* or the sublime *ifonly*? Do write!

Liverpool-born Stan Kelly-Bootle has been computing, on and off, at most levels since the pioneering EDSAC I days in the early 1950s at Cambridge University. After graduating from there in Pure Mathematics, he gained the world's first post-graduate diploma in Computer Science. He has authored "The Devil's DP Dictionary" and co-authored "Lern Yerself Scouse" and "The MC68000 Software Primer". ■

WHEN SERIOUS PROGRAMMING IS YOUR BUSINESS...

The Concurrent Euclid language for systems programming provides the best in efficiency, portability, reliability, and maintainability

Compilers running on UNIX/VAX, UNIX/11, VMS/VAX, with code generated for MC68000, MC6809, NS32000, 8086/8088 PDP-11, and soon running on IBM-PC

CONCURRENT EUCLID

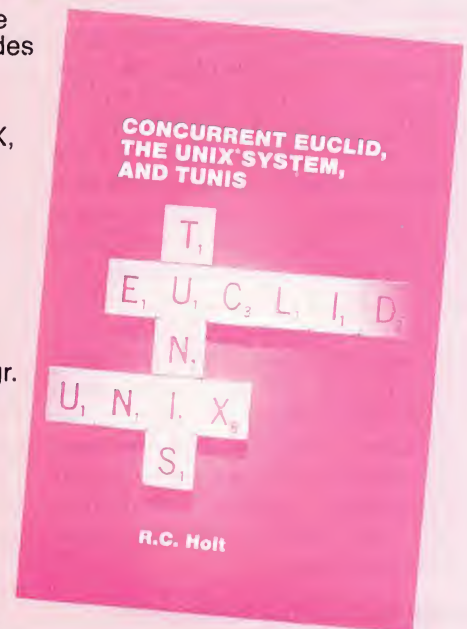
Compiler: CSRI Distribution Mgr.
Sandford Fleming Bldg 2002
10 King's College Road
Toronto, Canada M5S 1A4
Tel: (416) 978-6985

Book:

CONCURRENT EUCLID,
THE UNIX SYSTEM AND TUNIS

Available from:

Addison-Wesley Publishing
Company, Reading, MA. 01867
Tel: (617) 944-3700



CONCURRENT

E U C L I D

Circle No. 264 on Inquiry Card

PROBLEM SOLVER

Anatomy of a boot

by Bob Toxen

"Panic: init died!"

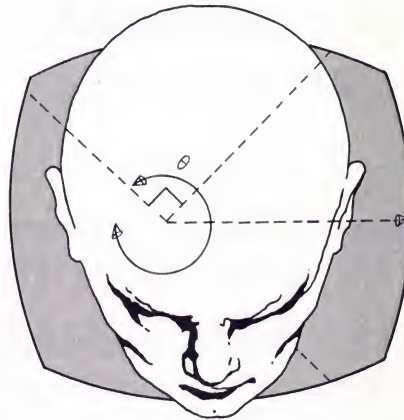
Most system administrators eventually see this message. It means that a system is dead or dying. If it occurs after the system has been running free of trouble for a while, it's indicative of a minor problem that can be solved by shutting down the system normally (if possible), or executing a **sync** and rebooting (running **fsck** in the process).

If, on the other hand, this same message or something similar appears when your system is booting up, you *should* panic! It means that files critical to your system's operation are incorrect or missing. In a previous issue (October, 1984), we investigated what a system administrator can do to prepare for this eventuality.

This month's column is concerned with what a system implementor (or anyone else with source code) can do to prevent the problem. I define the "system implementor" as a company that maintains system software. This is usually a hardware manufacturer.

THE BIRTH OF A KERNEL

In order to understand what prevents UNIX from booting up, one must first understand how it boots up normally. Many people know that to start a computer, they need only press a reset (boot)



button. Some UNIX systems even reboot automatically when you turn them on. This starts a program stored in non-erasable PROM memory called the "PROM monitor" or simply the "monitor".

This program in turn starts UNIX when a cryptic command is entered at the console terminal. The monitor then reads UNIX from the disk into memory and starts it running. Immediately, UNIX determines the amount of memory available in the system, ascertains how much is available for user processes, and displays the values on the console terminal.

If the system does not get to this point, it can be assumed that one of four things has happened. One, there may have been a hardware failure. Two, the hardware may have been incorrectly

configured; perhaps a DIP switch was accidentally bumped. Three, the wrong version of software may have been installed in either the PROM monitor or the UNIX kernel. Four, the copy of the kernel on disk may have been damaged or erased. The name of the file containing the kernel is usually */unix* or */vmunix*. A copy should be kept in a separate file as insurance against a damaged kernel. When */unix* (or */vmunix*) is changed, this backup copy should *not* be updated until after the new kernel has booted the system successfully. This is a hedge against the possibility that the new version will not work with your hardware or is otherwise defective.

THE KERNEL MATURES

After the kernel has "sized memory", it initializes any hardware needed for the root and swap disk devices. (Initialization of the console *tty* device and memory already should have been performed by this point.) The kernel then simulates a **mount** system call to configure the root file system. Next, process zero (which will become the scheduler) is built and initiated. This process, which contains hand-compiled code copied from kernel data space, does a **fork** system call.

The child that is created,

T A N G O TM**Use Tango to:**

- Connect IBM and compatible PC's running DOS to UNIX systems.
- Offload processing to PC's.
- Control data and applications on remote PC's.
- Distribute processing between UNIX and PC's.

Buy Tango for:

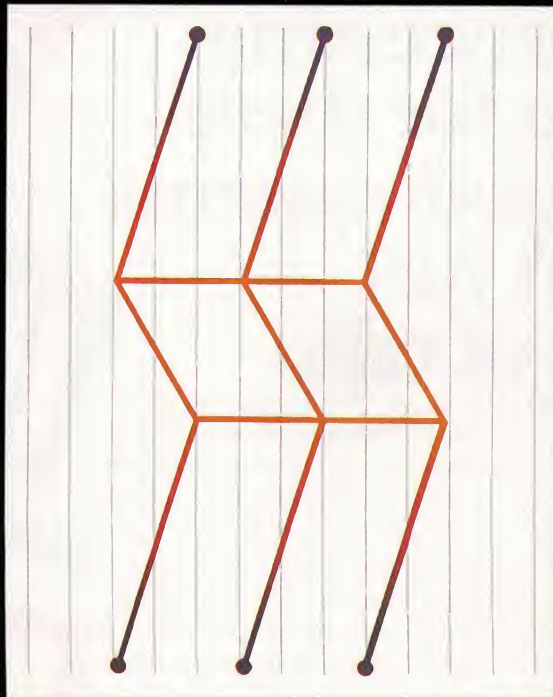
- Execution of DOS programs on the PC under UNIX control.
- Simple elegant file transfer under error correcting protocol.
- DEC, IBM, and Tektronix (graphics) terminal emulation.

Tango utilizes a standard RS-232 serial port on the PC and connects to the UNIX computer via a modem or direct connection.

COSI

313 N. First St.
Ann Arbor, Michigan
48103
(313) 665-8778
Telex: 466568

Tango is a trademark of COSI.
UNIX is a trademark of Bell Laboratories.



The PC-to-UNIX™ Connection

named process one, invokes an **exec** system call to start `/etc/init`. The parent, process zero, then becomes the scheduler, also known as the *swapper*. This is not a user process but rather just another face of the kernel itself. At this point, the kernel is fully operational.

If the **exec** of `/etc/init` fails (because `/etc/init` is missing or incorrect) or if **init** ever dies, the kernel will detect it and print the message "*panic: init died!*" In some implementations, though, this actually does not designate a *panic* situation (that is, a fatal error). Although the chance of a single file (`/etc/init`) getting damaged is small, the kernel can easily be modified to invoke, say, `/etc/getty` if **init** cannot be **ex-**

ec'd. Getty, like **init**, does not require standard input or output to be set up—unlike most other programs.

INIT FIRES UP (VERSION 7 AND BERKELEY UNIX)

Different versions of UNIX have different versions of **init**. On Version 7 and Berkeley UNIX, **init** forks off a child process that opens `/dev/console` for reading and writing. Since the system has had no open file descriptors up to this point in the startup procedure, `/dev/console` becomes file descriptor zero, which is also known as standard input. The **dup** system call is then invoked twice to duplicate this file descriptor for descriptors one and two, which are known as stan-

dard output and standard error. It then issues **ioctl** or **stty** system calls to set the correct baud rate, erase character, and so forth on the `tty` port. This child process then **execs** `/bin/sh` and *voilà*—the machine is in single-user mode.

INIT CHOKES AND DIES (VERSION 7 AND BERKELEY UNIX)

The kernel, `/etc/init`, `/dev/console`, and `/bin/sh` must all exist for the system to come up. A crash causing file system damage to one of these, or a problem as simple as an erroneous **chmod** can keep the system down for good. I have already covered contingency plans for the kernel and `/etc/init` being damaged. Let's

Only one word processing program for these UNIX[®]-based systems isn't just a lot of talk.

Many companies are promising UNIX-compatible word processing software. But only WordMARC[™] is being used successfully right now on such major UNIX-based systems as DEC,[®] HP,[®] SUN,[®] AT&T,[®] MASSCOMP,[®] PYRAMID[®] and NCR.[®]

With WordMARC, you'll have a single, full-featured program that will end the prolifera-

tion of word processing software. Training time will be cut because the *identical* program runs on all kinds of computers. So users can easily switch terminals or systems. And with its optional LinkMARC feature, text created on your UNIX-based system can be transferred to and shared by superminis and personal computers.

UNIX, DEC, HP, SUN, AT&T, MASSCOMP, PYRAMID and NCR are registered trademarks of, respectively, AT&T Bell Laboratories, Digital Equipment Corporation, Hewlett-Packard Co., Sun Microsystems, Inc., AT&T Bell Laboratories, Massachusetts Computer Company, Pyramid Technology Corporation and NCR Corporation.



now consider how to deal with */dev/console* problems. If either the **open** or **ioctl** system call fails, **init** can assume that the device node (the entry in */dev*) is bad.

To catch other problems, one also might set a 10-second alarm clock prior to an **open** call and turn it off when the **open** completes. This will account for situations where an **open** hangs, which may occur if the major or minor device values are wrong (they might, for instance, erroneously refer to a tape drive that already has been turned off).

If **init** determines that */dev/console* is bad, it can create its own version of the file. When **init** must resort to this, the file should be created in the root directory as a hedge against damage to the

The idea is to keep a tape of the shell and other useful programs around so they can be used when disaster strikes.

/dev directory. The file, typically called */console*, first should be removed with the **unlink** system call in case an old version exists, and then created with the **mknod** system call. The major and minor

device numbers that should be used will, of course, be hard-wired in **init** but these are unlikely to change from release to release and are usually both zero anyway. The **init** process then can open */console* instead of */dev/console*.

If the **exec** of */bin/sh* fails, it can try executing other programs that might allow the system to come up. If your system has **csch**, then */bin/csch* is a good second choice. It's possible that a copy of your shell also is kept in */etc*, so you might try to execute that file next. If this also fails, you can be assured that you have a very damaged file system. However, recovery is still possible.

One can create a copy of the tape (or floppy) device, usually

In addition to being compatible with all kinds of computers, WordMARC also gets along with all kinds of users.

Its documentation is written specifically for the computer system it will operate on. Its self-teaching guide helps novice users get quickly up to speed. And it's supported by a special "800" number hotline.

WordMARC's many versatile features include technical and scientific symbols, foreign language characters, a what-you-see-is-what-you-get screen, and menu-driven operation with

convenient function keys.

WordMARC can also be integrated with other popular applications software.

So get the UNIX-compatible word processing system that's up and running now—and put your word processing software resources back under control. With WordMARC. The Uncommon Denominator.

Contact MARC Software for more information. 260 Sheridan Avenue, Suite 200, Palo Alto, California, 94306.



MARC SOFTWARE INTERNATIONAL, INC.
1-800-831-2400. In California 1-800-437-9900.

Circle No. 260 on Inquiry Card

WordMARC
The Uncommon Denominator

WordMARC is a trademark of MARC SOFTWARE INTERNATIONAL, INC. © 1985, MSI, INC.



`/dev/rmt0`, in the root directory in much the same way as a copy of the console was created. A temporary file, say, `/tmpexec`—with mode 777—can then be created. This will allow `init` to copy data from the tape drive to the temporary file until an EOF is reached on the tape. The `init` process can then close both file descriptors, issue a `sync` system call, sleep for 10 seconds, and `exec /tmpexec`. The idea is to keep a tape of the shell and other useful programs around so they can be used when disaster strikes. The material on this backup can then be loaded into the system and used to fix damage. It may be necessary to create special versions of the utilities to be included in the

**An insurance policy
you should keep in the
vault is a recent backup
of all files.**

backup since the loading procedure will not allow arguments to be supplied. The `tar` and `fsck` commands are likely candidates for such modification.

Another problem to deal with is that the single-user shell may be successfully `exec'd` but then die

immediately thereafter. This can happen if part of the binary gets clobbered in such a way that it starts up but quickly core dumps. A way to detect this is to have the parent process invoke the `time` system call before the `fork` occurs (prior to `execing` the child process) and then check the `wait` afterwards to see how long the child was alive. If it was less than roughly 15 seconds, the shell can be assumed to have terminated abnormally and the parent, `init`, should be prodded into invoking other programs such as `csch` or `tar`, or possibly into using `/console` instead of `/dev/console`.

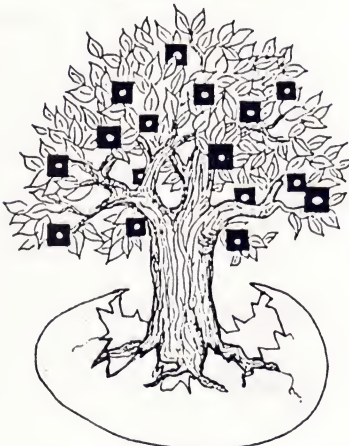
There are other possible techniques. One is to create a file system on a floppy (or a tape, if you are clever), including such critical programs as `sh`, `ls`, `tar`, `chmod`, and so forth. The `init` process could then attempt to mount the floppy when an `exec` of `/bin/sh` fails. In some cases, it may turn out that the best alternative is simply to fix the hardware and reload lost software from backup media.

**INIT GOES MULTIUSER
(VERSION 7 AND BERKELEY
UNIX)**

If all goes well, the parent will see its child process die. The final blow is usually delivered by a CTRL-D. The parent, `init`, then enters multiuser mode. This means that it reads the `/etc/tty` file and `forks` and `execs` a `getty` (`/etc/getty`) for each `tty` that users will be allowed to login at. Each `getty` opens a `tty` device specified in `/etc/tty` for standard input, output, and error, and then prompts for a login name. It then `execs` `login`, using the login name it receives as an argument. `Login` goes on to prompt for a password, verify it against `/etc/passwd`, and start whatever shell it finds listed in `/etc/passwd`.

Tree Shell

A Graphic Visual
Shell for Unix/
Xenix End-Users and
Experts Alike!



Dealer inquiries welcomed.

COGITATE

"A Higher Form of Software"
24000 Telegraph Road
Southfield, MI 48034
(313) 352-2345
TELEX: 386581 COGITATE USA

Circle No. 262 on Inquiry Card

"HOT" OPPORTUNITIES

CAMBRIDGE-BOSTON RT. 128

- UNIX/LISP
- GRAPHICS/PIXELS
- UNIX KERNAL
- MS/DOS SPREADSHEET
- X.25 LAN/COMMUNICATIONS
- HARDWARE DEVELOPMENT
- "C" Application Dev.
- MICROCODE 3D
- R&D Equity Positions

*Positions are all levels of expertise and numerous.

• CALL NOW (617) 868-5188
JIM BARRY



or send your resume
**ADVANCED ERGONOMIC
MANAGEMENT**

Software & Engineering
Personnel Consultants
18 BRATTLE ST. #451
CAMBRIDGE, MA 02138

Circle No. 261 on Inquiry Card

INIT (SYSTEM III AND SYSTEM V)

In System III and V, the administrator is given more control over **init** states (generically called single-user and multiuser modes) by configuring the ASCII file */etc/inittab*. Under System III, one specifies the program that should be invoked on particular *ttys* in certain states. State 1 is considered to be single-user mode and one usually starts */bin/sh* or */bin/csh* on */dev/console*. For additional security, one might wish to invoke **login** instead.

Under System V, single-user mode is called "state s". When this state is entered, **init** will first look in the */etc/inittab* file to see if it should enter single-user mode or one of the multiuser modes when the system is first booted. If single-user mode is specified (with the *defaultboot* entry, or simply by default), **init** will invoke **su** which in turn will look in the */etc/passwd* file for an entry called *root*. The **su** command will then **exec** the program specified in this entry as the shell. Thus, if */unix*, */etc/init*, */etc/inittab*, */bin/su*, */etc/passwd*, or */bin/sh* (or */bin/csh*) is damaged, the system will not be able to come up. This should illustrate the perils of requiring so many files to exist and be correct for a system to initialize correctly. The dangers are even greater than they might initially appear because the administrator will frequently have cause to alter */etc/passwd* and */etc/inittab*. The **init** program can be modified to deal with these problems by using the techniques discussed in "Init Chokes and Dies (Version 7 & Berkeley UNIX)".

Be on guard, though—if the */etc/inittab* file is missing, the System V **init** program still will prompt the user on the console for the correct state to enter but due to a bug, it will not accept

data that has been specified using a computer based on the MC68000. The bug makes **init** dependent on the byte ordering of **ints**. To cure the bug, search for where **init** attempts to read a single byte into the variable *c*, which is declared as an **int**. Use a variable declared as a **char** instead in these instances. If you should decide, though, to add these features to the kernel and **init**, be sure you have a way to boot your system from a different disk whenever you debug your code!

I have implemented most of the features described here and thus have been able to boot up many systems and remedy many problems that otherwise would have been untouchable. These steps

should prove to be good insurance for you as well.

Another insurance policy you should keep in the vault is a recent backup of all files. Under no circumstances should the steps proposed in this article be considered as a replacement for regular backup procedures; consider them, rather, as a complement. With a full backup to resort to, you'll be able to restore vital system and user files even after the severest disaster.

Bob Toxen has gained a reputation as a leading expert on UUCP communications, file system repair, and UNIX utilities. He has also done ports of System III and System V to systems based on the Zilog 8000 and Motorola 68010 chips. ■

UNIX/XENIX Communications Available NOW!

Put your computers on speaking terms.



Introducing **TERM.** Communications Software

Everyone from the beginning computer user to the expert finds TERM easy to learn and powerful to use. Just plug it in and go! In a few keystrokes you can access a remote database or send a group of files to another system.

TERM allows your computer to perform efficient, error-free exchange of binary or text files, over phone lines or hard-wired circuits at speeds of up to 9600 baud. Available options allow you to include or exclude a group of files for transfer in a single command.

TERM's "data capture" feature allows saving transcripts of sessions with remote mainframe and minicomputers to disk for later editing or printout, if desired.

- Pre-installed and ready to run
- Automatic error checking and re-transmission
- Wildcard (*) file send/receive capability
- Xon/Xoff, Etx/Ack, Ascii protocols for communications with non-TERM systems
- Full/half duplex emulation mode for remote systems
- Modem7 protocol for remote bulletin boards
- Auto-dial/Answer and Hangup supported on Hayes Smartmodem 300/1200 and compatibles
- Programmable batch file capability
- Unattended file transfer/auto logon
- Translation tables for input and output
- Remote maintenance capability

Term is available NOW on the Altos 586, IBM AT, Tandy Model 16, AT&T 362 and IBM PC/XT, MSDOS and many others. Find out how easy it is to get your UNIX, Xenix, and MSDOS machines all talking together



CENTURY
SOFTWARE

We make it easy for you.

9558 South Pinedale Circle
Sandy, Utah 84092
(801) 943-8386

Circle No. 263 on Inquiry Card

THE UNIX GLOSSARY

Language of languages

by Steve Rosenthal

Note: only those meanings related to computer languages are included in this installment.

assembler—a program which translates computer instructions from symbolic form to the machine language suitable for execution by a computer. "Assembler" is also sometimes used loosely to refer to input "assembly language". Although strictly speaking, an assembler translates each assembly language source statement into one machine code, more complex programs called "macro assemblers" can translate one source statement into many machine codes. The standard UNIX assembler is usually called **as**, but the actual program varies with the hardware it runs on. Many UNIX compilers (which translate high level language to machine code) actually emit assembly language as their output, relying on **as** to assemble the code to executable form.

axiomatic—said of languages that are expressed in terms of explicitly defined rules, or in terms of the logical extension and combination of those rules. In the UNIX community, Pascal and Modula-2 are most often used as examples. C, by contrast, is not axiomatic because its general rules are broken by large numbers of special cases and exceptions.



binding—refers to a connection between a language and an external set of routines or resources, such as a graphics package. Producing an error-free binding between an external resource and an existing language is often a difficult programming task.

cast—to change a value expressed in one data type or format to an equivalent value in another format. Some languages, such as C, are relatively tolerant of casts and even do some automatically. Others, with strong typing, such as Pascal, allow them more grudgingly, and even then only through explicit functions.

compiler—a software package that converts a complete program or module from a high-level language used by people into machine language instructions that a computer can actually execute. Once the compiled program is

saved in machine language form, it can be used again without retranslation. Developing a compiled program is more difficult than writing one for an interpreter (where the program is translated each time it is run), but compiled programs run faster.

compile-time error—a mistake flagged by a compiler program written during the process of translating a program in a high-level language to executable form. Because in most cases the compiler can point accurately to the offending statement, compile time errors are often easier to fix than those of the more elusive runtime variety.

context-free—said of languages and their grammars (or syntax) where words or symbols can be analyzed or substituted for without consideration for adjacent elements. Context-free grammars are easy to process, but they're further from human language use than grammars that do consider context.

control structure—in a computer language, the constructs that direct the flow of a program from statement to statement. Most theorists consider that there are less than half a dozen distinct control structures. In the programming philosophy known as structured programming, control structures that encourage modularity and program readability are

encouraged, while **goto** statements and other forms of uncontrolled transfer are discouraged.

high-level language—a computer language for writing programs in which the statements connote logical operations rather than exact steps for a machine to follow. A single high-level language instruction may cause a machine to execute tens or even hundreds of machine instructions. High-level languages must be translated into machine language before a computer can execute them. This task is handled by compilers and interpreters. C and the UNIX shell are high-level languages, but C also has features that allow low-level interaction with the hardware.

high-order language—another term for "high-level language" that's used mostly in Europe and by academic experts in this country. See *high-level language*.

incremental compiler—a program that accepts statements in a high-level language and translates them into an executable or intermediate form without waiting for the collection of all input statements. Incremental compilers thus offer the advantage of immediate feedback (like interpreters) and the fast running times characteristic of compilers. Several proprietary incremental compilers have been developed for C and other languages running under variations of UNIX.

interpreter—a software package that translates a program from a high-level language to executable form by translating and executing each line in turn without waiting to translate the program as a whole. Interpreters make it easy to write and debug programs since projects can be built up from small parts and tested easily, but interpreters require that time be taken to re-translate

programs every time they are run, even if there are no errors. Standard UNIX is compiler-oriented, but interpreters are offered for some languages in many commercial implementations.

lexical—referring to words or their formation. Most computer languages have lexical rules specifying reserved words, symbol sets, the construction of names, the treatment of case, and so on. These are supplemented by the syntactical rules that determine how words can be put together in statements.

low-level—said of computer languages or operations that are expressed in terms closely related to hardware rather than more general logical abstractions. Low-level languages (such as assembly language) allow more control and faster program execution, but they are more difficult to write and debug. One reason for the success of C as a language has been that it is basically a high-level language that nevertheless allows some degree of low-level programming. Consequently, even much of the UNIX kernel is now written in C.

meta-language—a formal symbolism used to describe a computer (or natural human) language. The meta-language most commonly used is BNF (Backus-Naur Form), which makes extensive use of brackets, braces and capitalization (or boldface) to show which elements are required or optional parts of statements and which are descriptions or explanations.

Modula-2—a language based on Pascal that offers improvements and additions that make it more suitable for production use. Like Pascal, it was written by Niklaus Wirth. Some UNIX programmers feel that because Modula-2 has many of the best features of both

Excellent Dealer Opportunity...

LIONS GATE SOFTWARE has completed the conversion and implementation of the complete **MCBA** library of packaged software products for operation on most computers.

Features of this unique set of software solutions include:

- Portability** – MS/DOS, XENIX and UNIX V
- Performance** – Benchmark Tests
- Support** – Full Warranty Service including 800# Telephone Support
- Marketing Assistance** – Customized Sales Brochures
- Aggressive Pricing**
- Complete Documentation** – Excellent User and System Manuals

Your business will be based on **15 Fully-Integrated Accounting, Distribution, Manufacturing and Report-Writing products.**

Inquire today!

In the U.S.A.: (800) 663-8354

Elsewhere: (604) 437-0001

We now have demonstration and source programs available for immediate delivery on the **AT&T 6300, 7300 and 3B series.**



LIONS GATE SOFTWARE INC.

2555 Gilmore Avenue,
Burnaby, B.C. Canada, V5C 4T6

Please send me information on this **excellent dealer opportunity:**

Name _____

Phone Number () _____

Company _____

Address _____

City _____

State _____ Zip Code _____

(End User Inquiries Welcomed)

Circle No. 257 on Inquiry Card

adax

X.25 FOR UNIX* Communications System

- Efficient, error-free data transmission to multiple hosts via international standard X.25, the only fully certified error-free public networking system used world-wide.
- User utilities
 - Remote user login
 - Remote mail service
 - Remote file transfer
- Compatible with widest number of host computers.
- Hardware available for VME, Multibus and others.
- Previously certified on TELENET, TYMNET and UNINET networks.
- Lowest cost per node.

Adax, Inc.

737 Dwight Way
Berkeley, CA 94710
(415) 548-7047

* UNIX is a trademark of Bell Laboratories.

Circle No. 259 on Inquiry Card

GLOSSARY

Pascal and C, it should supplant the latter as the system language for UNIX programming.

modularity—the degree to which a computer language supports the decomposition of processes into smaller units with clearly defined interfaces and interactions. C tolerates but does not especially encourage modular programming, while Pascal virtually demands it. Modular programs are easier to understand and maintain, but they also take extra time to produce.

non-procedural language — a computer language that expects the user to provide a description of the available input data, the desired output, and their relation, but lets the system choose the steps needed to produce the output. Some UNIX databases and program generators employ non-procedural languages, and artificial intelligence (AI) research promises to extend this approach to other areas as well.

object-oriented language — a language based on descriptions of logical objects, each of which can be acted upon by a data structure and a set of valid operations. Data is stored as instances of objects, which can interact by sending and receiving messages. One primary advantage of this approach is that each object can be treated as a "black box", with details of its operation hidden from other objects. This, in turn, allows for easier maintenance and upgrading.

pre-processor — a software package or part of a compiler program that translates a special dialect or abbreviated form of a computer language into a standard format for further processing. In UNIX, the RATFOR (Rational Fortran) pre-processor that translates RATFOR statements into Fortran is the best known, but the standard C compiler also

includes a pre-processor stage that expands macros and does other symbolic manipulation.

procedural language—a computer language in which the user specifies the flow of control and the operations to be performed on data. Most popular computer languages used with UNIX are procedural, including C, Fortran, BASIC, Pascal and Modula-2.

runtime error—an error that occurs during the execution of a program rather than during the translation of the program from high-level language to executable form. Runtime errors can cause program crashes, or, worse yet, erroneous results that offer no explicit warnings.

runtime error checking—a facility offered by some languages, notably Pascal and related dialects, that compares the value of variables (and sometimes program flow) against their defined possibilities. Runtime error checking does slow program execution, but it can be an important aid in the war against logical and notational errors. Unfortunately, most implementations of C under UNIX do not offer this option.

self-documenting—said of computer languages that are written in a form that can be easily read and understood, allowing programs to be written without an abundance of explicit comments and explanations. C, because of its terseness, is not normally considered self-documenting, while Pascal, COBOL, and Modula-2 often are. In reality, however, extensive comments are usually required even in the latter group of languages to make all but the simplest programs comprehensible and maintainable.

semantic—of or referring to the meaning of words or statements. At this stage, most computer

language processing is based purely on syntactical and lexical analysis, but one promise of artificial intelligence (AI) technology is that someday semantic factors will be considered as well.

separate compilation—translation of a portion of a program from source language to executable form without recompiling the rest of the program. Some compilers, including most of those for Pascal under UNIX, offer this capability. By allowing modules that are compiled separately to be combined and run together, separate compilation encourages modular programming and a team approach to large projects.

strong typing—the requirement that variables be defined according to the type of data they will contain (and, optionally, according to the valid range they'll fall in) before they're actually used in a program. Languages in the Pascal family use strong typing, but C and BASIC do not.

syntax—the rules for combining words and symbols to create valid statements in a given language. Also known as the "grammar" of the language.

terse—said of a language that requires few words or symbols to express operations. Both C and the UNIX shell language are terse, for example, while COBOL is not. Terseness is valued by hackers and frequent users, but the corresponding obscurity is often a problem for occasional users or people who must maintain other people's code.

threaded language—a language that allows users to define new commands with existing terms, and then use the new commands both to execute programs and define further commands. During processing, the program is treated as a linked list, with the system following terms back up

the list until it reaches a core vocabulary. FORTH is the best known of the threaded languages.

Turing complete—said of a computer language that can express all possible computational operations (although not necessarily efficiently or elegantly). All of the general-purpose languages for UNIX are Turing complete, but some of the more specialized formatting and text manipulation languages are not. The term is a reference to the Turing Machine, a theoretical device developed by Alan Turing to show that all actual computer procedures can be modeled using a series of simpler operations.

type—refers to the variety of data that a variable, constant, or function can contain or produce. Most computer languages support a range of simple data types (such as integers, characters, and real numbers) as well as composite types (such as arrays and records). UNIX has traditionally favored languages with weak typing, such as C and BASIC, but strong-typing languages such as Pascal and Modula-2 are garnering increased interest due to their better readability and maintainability.

verbose—said of a language that uses many words or symbols to express an operation. COBOL is probably the example most often cited. While many UNIX people prefer terse languages such as C, verbose languages tend to lend themselves to readable programs and easy program maintenance.

Comments, questions, corrections? Please send them to Rosenthal's UNIX Glossary, Box 9291, Berkeley, CA 94709.

Steve Rosenthal is a lexicographer and writer living in Berkeley. His columns regularly appear in six microcomputer magazines. ■

CEEGEN-GKS GRAPHICS SOFTWARE in C for UNIX

- Full implementation of Level 2B GKS.
- Outputs, Inputs, Segments, Metafile.
- Full Simulation for Linetypes, Linewidths, Fill Areas, Hatching.
- Circles and Arcs, Ellipses and Elliptic Arcs, Bezier Curves.
- Ports Available on all Versions of UNIX.
- CEEGEN-GKS is Ported to Gould, Masscomp, Plexus, Honeywell, Cadmus, Heurikon, Codata, NBI, NEC APCIII, IBM-AT, Silicon Graphics, Pyramid, Tadpole Technology, Apollo, AT&T 3B2, AT&T 6300, DEC VAX 11/750, 11/780 (4.2, 5.2), NCR Tower.
- CEEGEN-GMS GRAPHIC MODELING SYSTEM: An Interactive Object-Oriented Modeling Product for Developers of GKS Applications. CEEGEN-GMS and GKS Provide the Richest Development Environment Available on UNIX Systems.
- Extensive List of Peripheral Device Drivers Including Tektronix 4010, 4014, 4105, 4109, HPGL Plotters, Houston Instruments, Digitizers, Dot Matrix Printers and Graphics CRT Controllers.
- END USER, OEM, DISTRIBUTOR DISCOUNTS AVAILABLE.



CEEGEN CORPORATION
20 S. Santa Cruz Avenue, Suite 102
Los Gatos, CA 95030
(408) 354-8841
TLX 287561 mlbx ur

EAST COAST
John Redding & Associates
(617) 263-8206

UNITED KINGDOM
Tadpole Technology Pl C
044 (0223) 861112

UNIX is a trademark of Bell Labs.
CEEGEN GKS is a trademark of
Ceegen Corp.

Circle No. 258 on Inquiry Card

RECENT RELEASES

ADA GOES ABROAD

Verdix Corp. has signed an agreement with GEC Software Limited, a wholly-owned subsidiary of the General Electric Company PLC, Britain's largest electrical and electronics company. GEC will become the European center for sales, support, and training of all Verdix Ada and Ada-related products. Meanwhile, Ada products developed by GEC will be marketed in the US through Verdix. The agreement also gives GEC marketing rights for the family of Verdix Ada products in all non-communist European countries, Australia, and New Zealand.

Ada, developed by and known for its use in the US Department of Defense, now has an opportunity for growth within the European defense industry. According to Derek Alway, Managing Director of GEC Software, "Amendments have been made to the US Code of Federal Regulations ...which re-classifies software as technical data, thus removing the requirement to obtain a validated export license for export of certain software, including Ada software."

The agreement with GEC follows two recent Verdix agreements with US defense and aerospace contractors, Martin Marietta and Honeywell. Martin Marietta has acquired a 16 percent equity interest in Verdix, and Honeywell has signed a major End User License Agreement to use the Verdix Ada Development System (VADS) in the development of Ada software for embedded systems.

Currently VADS, including the Verdix Ada compiler, runs under 4.2BSD and Ultrix on the range of DEC VAXen from the 11/730 upward. Verdix also recently ported its VADS to Sun Microsystems' 68000-based workstation.

Verdix Corp., Westgate Research Park, 7655 Old Springhouse Rd., McLean, VA 22102. 703/448-1980.

Circle No. 224 on Inquiry Card

FORTRAN ANYONE?

A prime focus of IBM's plans for the PC AT is to penetrate the scientific applications market. With this in mind, Ryan-McFarland Corp. is preparing an implementation of its RM/Fortran compiler to run under Xenix on the PC AT and other 80286/AT-compatible systems. RM/Fortran is the same compiler IBM markets as IBM PC Professional Fortran for its Engineering and Scientific Series of machines. The new Xenix implementation can execute the same source code developed for the DOS implementation.

The Xenix 286 product has certain requirements, however—namely, an 80287 floating point processor (selling for around \$200), the Xenix system, and the Xenix Software Development System. RM/Fortran is, however, a complete implementation of the Fortran-77 standard, certified as error-free by the GSA. It also supports arrays and programs larger than 64 K bytes, and has extensions and features found in mainframe applications, including symbolic names of up to

31 characters, Hollerith and hexadecimal constants, and Industrial Real-Time Fortran (IRTF) binary pattern and bit-processing functions.

The PC AT and the Altos 2086 are the first machines on which the new version, due out this month, will be available. Suggested retail for Xenix RM/Fortran is \$750.

Ryan-McFarland Corp., 609 Deep Valley Dr., Rolling Hills Estates, CA 90274. 213/541-4828.

Circle No. 225 on Inquiry Card

MAKING OPTIMUM USE OF MEMORY

The Optimum 1000, the first optical disk drive announced by an American company a little under two years ago, is slowly working its way into use with UNIX-based systems. Any product that makes large amounts of storage more accessible is worthwhile to UNIX users, and the 1000, which stores 1 gigabyte of data on one side of a single 12-inch removable disk using write-once technology, could be a good answer.

Upgraded to be mode-compatible with 3M media, the Optimum 1000 is now being used on two UNIX-based systems, according to Larry Fujitani, Optimum Director of Marketing. One of these, a MassComp machine, was ported by a system integrator in Seattle, and the other is under development. More ports to UNIX systems are being considered by Optimum.

The single-quantity price for



Lifeboat.™

C is the language.
Lifeboat™ is the source.

Productivity Tools from the Leading Publisher of C Programs.™

The Lattice® C Compiler

The cornerstone of a program is its compiler; it can make the difference between a good program and a great one. The Lattice C compiler features:

- Full compatibility with Kernighan and Ritchie's standards
- Four memory model options for control and versatility
- Automatic sensing and use of the 8087 math chip
- Choose from the widest selection of add-on options
- Renowned for speed and code quality
- Superior quality documentation

"Lattice C produces remarkable code...the documentation sets such a high standard that others don't even come close...in the top category for its quick compilation and execution time and consistent reliability."

Ralph A. Phraner, *Byte Magazine*

Lattice Library source code also available.

Language Utilities

Pfix 86/Pfix 86 Plus — dynamic and symbolic debuggers respectively, these provide multiple-window debugging with breakpointing capability.

Plink 86 — a two-pass overlay linkage editor that helps solve memory problems.

Text Management Utilities — includes GREP (searches files for patterns), DIFF (differential text file comparator), and more.

LMK (UNIX "make") — automates the construction of large multi-module products.

Curses — lets you write programs with full screen output transportable among all UNIX, XENIX and PC-DOS systems without changing your source code.

BASTOC — translates MBASIC or CBASIC source code directly to Lattice C source code.

C Cross Reference Generator — examines your

C source modules and produces a listing of each symbol and where it is referenced.

Editors

Pmate — a customizable full screen text editor featuring its own powerful macro command language.

ES/P for C — C program entry with automatic syntax checking and formatting.

VEDIT — an easy-to-use word processor for use with V-PRINT.

V-PRINT — a print formatting companion for VEDIT.

CVUE — a full-screen editor that offers an easy way to use command structure.

EMACS — a full screen multi window text editor.

Fast/C — speeds up the cycle of edit-compile-debug-edit-recompile.

Graphics and Screen Design

HALO — one of the industry's standard graphics development packages. Over 150 graphics commands including line, arc, box, circle and ellipse primitives. The **10 Fontpack** is also available.

Panel — a screen formatter and data entry aid.

Lattice Window — a library of subroutines allowing design of windows.

Functions

C-Food Smorgasbord — a tasty selection of utility functions for Lattice C programmers; includes a binary coded decimal arithmetic package, level 0 I/O functions, a Terminal Independence Package, and more.

Float-87 — supports the 8087 math chip to boost the speed of floating-point calculations.

The Greenleaf Functions — a comprehensive library of over 200 routines.

The Greenleaf Comm Library — an easy-to-

use asynchronous communications library.
C Power Packs — sets of functions useful for a wide variety of applications.

BASIC C — This library is a simple bridge from IBM BASIC to C.

Database Record Managers

Phact — a database record manager library of C language functions, used in the creation and manipulation of large and small databases.

Btrieve — a sophisticated file management system designed for developing applications under PC-DOS. Data can be instantly retrieved by key value.

FABS — a Fast Access Btree Structure function library designed for rapid, keyed access to data files using multipath structures.

Autosort — a fast sort/merge utility.

Lattice dB-C ISAM — a library of C functions that enables you to create and access dBase format database files.

Cross-Compilers

For programmers active in both micro and mini environments we provide advanced cross-compilers which product Intel 8086 object modules. All were developed to be as functional — and reliable — as the native compilers. They are available for the following systems:

VAX/VMS, VAX/UNIX, 68K/UNIX-S,
68K/UNIX-L

Also, we have available:

Z80 Cross-Compiler for MS- and PC-DOS — produces Z80 object modules in the Microsoft relocatable format.

New Products

Run/C — finally, a C interpreter for all levels of C Programmers.

C Sprite — a symbolic debugger with breakpoint capability.

Call LIFEBOAT: 1-800-847-7078. In NY, 1-212-860-0300.

YES! Please rush me the latest FREE Lifeboat™ catalog of C products.

Name _____ Title _____

Company Name _____ Business Phone _____

Address _____

Please check one of the following categories:

Dealer/Distributor End User Other _____

Return Coupon to: Lifeboat™ Associates
1651 Third Avenue, New York, NY 10128

Circle No. 252 on Inquiry Card

© 1985 Lifeboat Associates





The Optitem 1000 optical disk drive.

the Optitem 1000, with SCSI interface, is \$13,600.

Optitem, 435 Oakmead Parkway, Sunnyvale, CA 94086. 408/737-7373.

Circle No. 242 on Inquiry Card

**COMPUTER WARRANTY
PARALLELS CHRYSLER**

Perhaps finding inspiration in Lee Iacocca, Parallel Computers has announced two new models

of minicomputers with an optional warranty covering the cost of all maintenance for five years (but not 50,000 miles).

The Parallel 300 XR Model 30 and Model 40, replacing the company's 300 system family, are based on redundant, self-checking architecture. All key components, including CPU, memory, disk subsystems, and power supplies (with integrated, uninterruptible power systems), are duplicated. If a component fails, its twin maintains operation.

The Model 30 is made to support eight users with 1 MB of main memory, 84 MB of hard disk, and a 1/4-inch streaming tape. The Model 40 supports up to 16 users with 2 MB of main memory and 168 MB of hard disk storage. Both machines are based on a 68010 processor running at 10 MHz, with Multibus. A high-end configuration of either machine allows support of up to 32 users with 8 MB of main memory and 2-plus GB of disk storage.

Regarding maintenance, Brian Knowles, Parallel's Director of Marketing, outlines three stages of repair should the machine need it. The first is built into the box itself—both the 30 and 40 provide automatic fault detection messages to the user. Second, remote diagnostics can be performed from Parallel headquarters. Finally, Parallel technicians can come to the field site.

Both the 30 and 40 are available now. The base configuration price for the Model 30 is \$59,900; for the Model 40, \$74,900. A typical high-end configuration, depending on options, is priced in the \$80-100,000 range. Given the traditional cost of maintenance for micros, one may consider purchasing the five-year warranty for \$9000. Though Parallel's 300 system family is being phased out, field upgrades are available for \$7000, and the up-

FORTRAN 77

COMPILER INCLUDES FULL SUPPORT FOR MOTOROLA'S

MC68020/68881

- Full ANSI 77 implementation
- Full Screen Source Level Symbolic Debugger
- Unix and C Interface (Unix is a trademark of AT & T)
- Generates 68000 and 68010 Code
- Support for NS32081 and SKY FFP Math Hardware

ALSO AVAILABLE 68020/68881 MACRO ASSEMBLER

- 100% Motorola Compatible - Includes C Interface
- 2X to 20X Faster Than Most Assemblers

abs:ft

**SCIENTIFIC/ENGINEERING
SOFTWARE** 4268 N. Woodward
Royal Oak, Michigan 48072
(313) 549-7111 • TX 235608

LANSCAPES: Local Area Network Performance Profiles

**FREE Software
with each order.**

The rush to install local area networks is on. Corporate data processing professionals, system integrators, OEMs, network software developers and LAN companies themselves are faced with a bewildering array of network options for personal computers. The need is there to connect PCs into an effective communications system, but can the available products meet the criteria and specifications demanded by both users and systems houses?

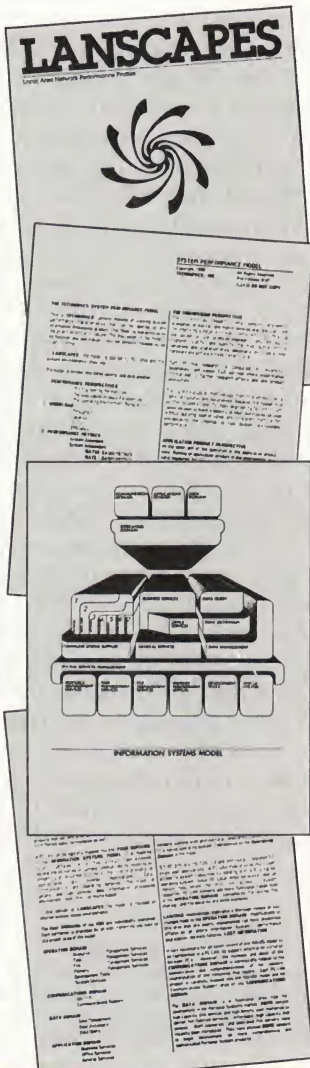
Managers now have an easy-to-use information source to evaluate whether a network will perform as promised: LANSCAPES—Local Area Network Performance Profiles. LANSCAPES is a formalized measurement and evaluation method which models performance characteristics of local area networks designed to link IBM PCs and/or compatibles.

CONTENTS

This exclusive new management report, LANSCAPES, evaluates the performance of network products from Fox Research (10-Net), Orchid Technology (PCnet) AST RESEARCH (PCnet II), 3COM (EtherSeries & 3Server), Corvus Systems (Omnishare & Omninet) and the IBM PC Network. The performance evaluations were conducted by the independent information systems research company, Cronotec, Inc. of La Jolla, Calif. Cronotec's Local Area Network Metric Analysis System (LAN/MAS) maps PC LAN products into two distinct, but complementary models. Its *Information Systems Model* provides a consistent reference for PC LAN physical and logical evaluation and the *System Performance Model* empirically portrays system behavior. The two models produce *Performance Profiles* which detail and measure true network performance. Each model and the entire testing and evaluation process is described in detail. Qualified source listings, a matrix of text numbers and bibliographic references are provided.

Information Systems Model: With this model, networks are mapped to the Operating Domain, the Communications Domain, the Data Domain and the Applications Domain. To be effective, a network must provide multi-user access to system resources by bonding with a PC's native operating system. To give users true multi-user ability, LAN vendors must add functional capabilities to the Operating Domain in the form of semaphores, file locking, file sharing and file security.

Cronotec's evaluations highlight a thorough review of key changes made to the Operating Domain to determine the impact on system performance. In the Communications Domain, LANSCAPES maps each network to the seven layers of the



ISO/OSI model. Network management, architecture and transmissions are detailed. The Data Domain maps the network data handling for organization, access, spontaneous and generalized query and control. The Applications Domain models performance for applications software running on the network. LANSCAPES maps the dependent functions of applications software to the communications capability of the network to model software performance.

System Performance Model: LANSCAPES measures network performance by throughput and capacity with two measurements—system dependent metrics and system independent metrics. Metrics are the count, time, rate and ratio measures of task sequences, and processing transactions. These selected series of task sequences in different environments and load scenarios yield actual throughput and capacity.

LAN Performance Profiles: To show the performance results of each LAN, the test results are described and then illustrated with tables, charts and graphs. Each network is measured against its own performance goals to find out if it meets its own specifications for data throughput and capacity.

FEATURES

Interpretative Text describes and defines the network evaluation environment for all configurations. Modeling techniques are illustrated to simulate a variety of network applications.

Graphic Illustrations for each network performance profile highlight key performance activities for selected processor tasks.

Separate Network Profiles: Each network evaluation is described in total, with pertinent mapping to the ISO/OSI seven-layer model.

FREE SOFTWARE WITH EACH REPORT

Included free of charge with each copy of the LANSCAPES is a powerful new software tool, CRONozap, Cronotec's Dynamic Memory Editor. The program comes on a 5-1/4" floppy diskette formatted for the IBM PC. This powerful new utility is used to view, print and/or modify main memory (RAM).

Because the screen is refreshed so quickly, it's possible to view the activity in memory dynamically, as it occurs—as though you were watching the computer "think." CRONozap can also be installed on a server to watch the data flow through the network buffers.

WHO SHOULD BUY LANSCAPES

LANSCAPES is intended for corporate data processing and MIS professionals, system integrators, OEMs, software engineers and others who are responsible for the design, implementation and modification of PC local area networks. Network manufacturers will also be able to provide third-party support vendors with this report as a scientific interpretation of network performance. The performance profiles were conducted at an independent laboratory where the networks were installed on an IBM PC/AT and four IBM PCs. Where a wrong decision in picking a local area network could lead to disaster, the 64-page LANSCAPES report at only \$197 is an investment that will repay its modest cost hundreds of times over.

ABOUT THE AUTHORS

Stephen L. Gubelmann has had 10 years of experience as a network designer. His positions have included network systems designer at CitiCorp's Transaction Technology Inc. and manager of network systems development at Home Federal Savings and Loan.

Robert Bennett has 17 years of technical, research and management experience in data processing for insurance, finance, health and manufacturing. His communications experience includes corporate networks and distributed systems.

Guaranteed No-Risk Offer

MI 8/85

MICRO COMMUNICATIONS Book Dept. • 500 Howard Street, San Francisco, CA 94105 • (415) 397-1881 • Telex: 278273

When ready in June, please send me _____ copies of LANSCAPES: Local Area Network Performance Profiles for my staff and associates at US\$197 each. Each copy will be shipped with the free bonus software program, Dynamic Memory Editor.

After seven days of reviewing the report, if for any reason I find LANSCAPES not acceptable or useful, I may return it for a prompt refund, and keep the bonus software, Dynamic Memory Editor, without charge.

Payment enclosed of US\$_____ for _____ copies. (Calif., Georgia, Ill. and N.Y. residents: please add sales tax.)

Charge: Visa MasterCard upon shipment.

Card No. _____ Expires _____

Signature _____ Date _____

Name _____ Title _____

Company _____

Address _____

City _____ State/Zip _____

grades are eligible for the warranty.

Parallel Computers, 3004 Mission St., Santa Cruz, CA 95060. 408/429-1338.

Circle No. 239 on Inquiry Card

GXL + VXL = ANN ARBOR

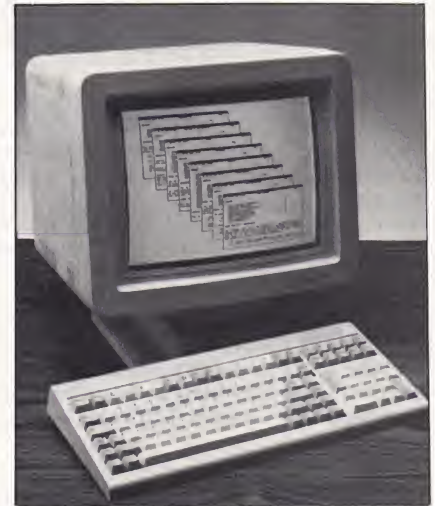
Two new VDTs now available from Ann Arbor Terminals come loaded with plenty of smarts.

The VXL is an ANSI standard, multi-host, multi-window terminal that is character-mapped for alphanumeric applications. Designed to accommodate the user who prefers one rather than several terminals on a desk, the VXL can work with up to four hosts simultaneously, with the ability to access IBM mainframes, DEC minis, and/or PCs. The user can switch from one host to another with a single keystroke, even while receiving results from other hosts in other windows.

Twenty kilobytes of local display memory are provided with the VXL, and this memory may be divided into up to eight pages, each of arbitrary width and

height—up to 255 columns and 512 lines. The user can dynamically connect (and disconnect) any page(s) to (from) any host(s), and dynamically switch the keyboard from host to host. Each VXL page is a virtual terminal: it can have its own setups and its own key programming to optimize it for efficient use with each host. The screen density is dynamically variable, from 80 to 160 columns and 36 to 60 lines. Users may work at whatever character size they find most comfortable, yet view and edit the contents of up to two 8-1/2 x 11 sheets of paper side by side when they wish. The VXL keyboard is fully programmable up to eight shift levels.

The Ambassador GXL+Plus offers full-page alphanumerics, raster-scan graphics, and a user-definable character set. It has the ability to transform mapping of a specified window in the drawing space to a specified region on the screen. The character set can be pre-set with any series of graphic instructions including alternate fonts, schematic symbols, or entire layers of graphics displays



The VXL from Ann Arbor Terminals.

that can be later recalled by pressing a key. An alternate Math and Greek character set is included for scientific applications.

Featured on the GXL+Plus is a resolution of 768 x 600 dot pixels, 4096 x 4096 addressable, on a 15-inch green phosphor screen. With Tektronix 4010/4014 compatibility and VT 640 enhancements, the terminal supports all popular graphics packages. Other standard graphics features include 11-line types for vector drawing, point and incremental plot modes, 16 polygon fill patterns, selective erase, and a mouse interface. The GXL+Plus comes standard with an 18-to-60-line-by-80-character display, two pages of memory, and full editing capabilities (offering block mode and form-filling functions). The keyboard is fully programmable on up to 32 levels. Transmission is character-by-character, line-by-line, or block mode with speeds ranging from 110 to 19.2 Kbps.

The list price of the VXL terminal is \$2795, and the GXL+Plus' list is \$3590. Ann Arbor sells to OEMs (including DEC and IBM for their machines running UNIX).

u4th

THE UNIX/XENIX-compatible Forth:

- C primitives
- dynamic memory management
- direct-threaded
- UNIX interfaces
- object-oriented Forth-source included!
- 400-page manual and glossary
- no royalties for developers

New Low Prices!
Plexus, Sun, Intel 286: \$495.00
PC XT, AT, Tandy: \$195.00
Now! VAX, AT&T 3B

UBIQUITOUS SYSTEMS
13333 BEL-RED ROAD NE
BELLEVUE, WA 98005
206-641-8030

UNIX: TM, AT, AT XENIX: TM, MICROSOFT

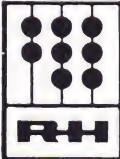
Circle No. 241 on Inquiry Card

EDPeople

SYSTEMS PROGRAMMER
Join this industry leader and continue to develop a wide range of applications. Your UNIX, C + assembly language is the key to the dynamic opportunity. Salary \$25-34K.

SR PROGRAMMER
Interested in developing, designing and maintaining new and current systems? One of the nation's largest "state of the art" data centers seeks highly promotable individuals interested in pushing the limits of the technology. CALL TODAY, if you have Mainframe, Micro, PLI, UNIX or C exp. Salary \$30-45K.

FEE PAID (513) 224-0600



ROBERT HALF
P.O. Box 756
Mid-City Station
28 N. Wilkinson Street
Dayton, Ohio 45402

®

Circle No. 240 on Inquiry Card

CONCENTRIC ASSOCIATES, INC.

WE DON'T PRODUCE TRAINING.

We Produce Shell Programmers, C Programmers, Ada Programmers, System Administrators, Kernel Hackers, Doc Preppies, and Project Managers.

- We will work with you to find out what your people need to know.
- At no charge, we will propose a curriculum tailored so that your people are immediately productive.
- Our instructors will deliver the courses or you can license the courses and we'll teach your teachers.

Circle No. 293 on Inquiry Card

WE ARE ALSO COMMITTED TO BRING TO MARKET A LINE OF SOFTWARE TOOLS TARGETED AT PROGRAMMER PRODUCTIVITY.

The first of these products is:

shacc—the **shell accelerator**—is a compiler for the Bourne shell. It translates Bourne shell programs into C and then invokes the C compiler to produce an "a.out" file. The C code that is generated is well-structured and very readable, so it can be further optimized by hand if you like.

shacc allows you to write production code in the Bourne shell: Do the fast prototyping in shell and then **shacc** it and ship it.

Call us for information about our on-line demonstration.

shacc
By Paul Ruel
Concentric Associates

SHACC UP WITH
CONCENTRIC
ASSOCIATES, INC.

For further information on our Educational Services or shacc, call or write:

Linda Cranston/Concentric Associates, Inc./One Harmon Plaza/Secaucus, NJ 07094
201-866-2880

See us at UNIX EXPO, New York, Booth #130

Circle No. 294 on Inquiry Card



Ann Arbor's GXL+Plus Terminal.

major customers, and end users. Ann Arbor Terminals, Inc., 6175 Jackson Rd., Ann Arbor, MI 48103. 313/663-8000.

Circle No. 236 on Inquiry Card

SUN IN THE BIG BLUE WITH MORNING STAR

For those with a Sun workstation and the desire to offload heavy calculations or hook up to a large information exchange

base, there is now a link to consider. Morning Star Technologies, whose customers include UNIX vendors MassComp, Pyramid, and Sperry, has announced that its UNIX communications products are now available to connect Sun workstations to IBM systems.

For the Sun-2/100U, 2/120, and 2/170 stations, Morning Star offerings include MST/X.25 and MST/HASP protocol software, which run on models of the Morning Star Horizon series communication processors. The communication systems are ready as drop-in products and include a Sun operating system device driver. The MST/X.25 packet switching protocol is Telenet and Uninet certified and complies with DDN recommendations; it sells for \$2995. The MST/HASP package emulates the full implementation of the IBM 360/model 20 RJE workstation; it sells for \$2400.

The two Horizon series communication processors, residing on boards that slip into the workstation and plug into Multibus (a product plugging into VME bus will be out soon, Product Support

Manager Jamey Laskey said), are based on a 68000 processor, have a minimum of 128K DRAM, and use the Morning Star Unidriver device driver. The Horizon Model 200 has two serial ports, runs at speeds up to 19.2 Kbps, and retails for \$1995. The Model 800 has eight serial ports, up to four channels of DMA, runs at 64+ Kbps, and retails for \$2395.

Also on the horizon, so to speak, are Morning Star products that include SNA/3270, SNA/3770, and Bisync 3270 and 3780; these are due out this autumn and will be compatible with current offerings.

Morning Star Technologies, 4510 Kenny Rd., Suite 204, Columbus, OH 43220. 614/451-1883.

Circle No. 235 on Inquiry Card

STAY WELL WITH INTEL

Intel Corp., increasing its vertical market activity beyond the finance and insurance industries, has announced a \$20 million, three-year volume agreement with Provider Automated Services, Inc. (PAS), a subsidiary of Blue Cross/Blue Shield of Florida. Under the agreement, the two companies will jointly develop supermicro systems for the health care industry.

The systems will be based on Intel's Multibus-based 286/310 supermicros running Xenix 3.0. The machines will be shipped to Jacksonville, where PAS's medical administration software package will be added. PAS is also responsible for marketing the product: its previous products are available in 39 states. The Intel-PAS systems are being released this month.

Provider Automated Services, Inc., 8659 Baypine Rd., Suite 200, Jacksonville, FL 32216. 904/739-6703.

Circle No. 234 on Inquiry Card

Save Time and Money on Data Entry

Use ZIPLIST to automatically look up city, state and county information based on zip code. Table of 48,000 zips allows significant savings on data entry, error corrections and file maintenance. This set of floppy disks, including easy instructions, is just \$149. Most popular 5¼" and 8" formats are available. Hard disk recommended. Call or write for free information.

DCC Data Service
1990 M Street, N.W. Suite 610
Washington, D.C. 20036

Call toll-free 1-800-431-2577
In DC & AK 202-452-1419

Circle No. 238 on Inquiry Card



Q'Nial

The new, sophisticated, interactive programming system for UNIX Workstations

NIAL Systems Inc.
1742 Second Ave, Suite 159,
New York, NY 10128
1-800-267-0660 (U.S.)

Q'Nial is a registered trademark of Queen's University at Kingston

Circle No. 237 on Inquiry Card

LANGUAGE TOOLS

Continued from Page 37

somewhat restricted and well understood domain to be successful, but they are very attractive when they make sense and can be implemented efficiently.

DOWNSTREAM

Lexical and syntactic analyses of languages are well understood, but what do we do once we have recognized a language? In many simple cases, no further tools are needed; the **yacc** actions can directly call functional routines in the application program, perhaps with arguments built up from **yacc** values.

In more complicated situations, the parser builds one or more data structures to represent the input, and then invokes routines to further process them. For example, in many traditional compilers, the parser generates both a parse tree (or other data structure) and calls to symbol table routines.

Because of the portability of the UNIX system, the language designer may have to write a language without knowing anything about the host machine on which it will run. There are two traditional approaches to this problem. One is to cause the compiler for the new language to generate some language that is already in common use on the target systems. For example, RATFOR and EFL generate Fortran, and C++ generates C. Several companies market compilers that will read Fortran or Pascal and convert it to C. This technique of *pre-processing* the language can be surprisingly difficult to do well, but it can attain the goals of portability and efficient output code.

Another technique is to take some compiler that is used on many different machines, and cause the new language to share the code generator with this compiler. For many years, Fortran and Pascal have shared code generators with the portable C compiler code generator, and there are other languages available that use code generators for Pascal p-code. While these techniques are less efficient than a custom-built compiler, and rarely are totally portable, they have proved easy to implement and useful in practice.

THE FUTURE OF LANGUAGES

Languages are used to communicate between computers and people. The UNIX command language style was a reaction to the chatty nature of some other operating systems ("are you sure you really want to quit now?") and the obscurity of others, such as OS/360. Now, systems such as the

Apple Macintosh have popularized a different way by which people can communicate with computers, using menus and icons. The decreasing cost of memory and processing power has made this an approach that is not much more costly than traditional ones. At the same time, smart editors have appeared that apply similar principles to the editing of more conventional languages. The AT&T UNIX PC, for instance, supports a menu interface to a number of administrative and office utilities, while still providing the experienced user with the means to use the power of the underlying UNIX system.

Will such interfaces eliminate languages as we know them? For some applications, I think it is clear that they will. Menus have a clear advantage for such sensitive operations as inserting new users into the system and doing file backup; the bit rate of human interaction is low, the operations are infrequent, and the consequences of error are serious.

Menu interfaces are restricted, however, in some ways that appear to be fundamental. How, for example, can we make a shell script out of icon and menu selections? (We face a similar problem making an editor script with **vi** or **emacs**.) We will always want to make new operations out of old, and build up complicated things out of simple ones. (Imagine a word processor where each word had to be picked by menu out of a dictionary, and then modified by menu to become either plural, past tense, or whatever. Typing is a skill that takes a while to learn, and it is prone to error, but the overall bit rate is much higher than what menus have to offer.) Because experienced programmers will always want terser commands with higher bit rates, the challenge is to retain the high efficiency and abstraction of current languages while capturing the safety and ease of newer techniques. Application designers certainly have their work cut out for them!

*Stephen Curtis Johnson has a BA from Haverford College and an MS and Ph.D. from Columbia University, all in Pure Mathematics. He has worked for AT&T Bell Labs since 1967. After early work in psychometrics, he has done research in computer algebra, parsing, complexity theory, code generation, portability of compilers and operating systems, and VLSI design. He is the author of a number of UNIX commands, including **yacc**, **lint**, the portable C compiler, and the first versions of **spell** and **at**. He is currently the head of the Language Development Department that provides computer languages for AT&T computers under System V.*

■

CALENDAR

EVENTS

SEPTEMBER

September 18-20 New York: "UNIX EXPO". Contact National Expositions, Don Bery, 14 W. 40th St., New York, NY 10018. 212/391-9111.

September 26-28 Boston: 8th Northeast Computer Faire, to be augmented with UNIX Systems Expo/85-Fall. Contact Computer Faire, Inc., 181 Wells Ave., Newton, MA 02159. 617/965-8350.

TRAINING

Note: Below are listed the dates, locations, titles, and contacts for UNIX-related training courses. For registration and further information on particular courses, contact the firm cited. Training firm addresses and phone numbers are listed alphabetically at the end of the calendar.

SEPTEMBER

September 2-3 London: "Shell Programming". Contact CTG.

September 3-5 New York: "UNIX Shell Programming". Contact LUCID.

September 3-6 Washington, DC: "UNIX for Users/UNIX Shell Programming". Contact USPDI.

September 4-6 Santa Monica, CA: "INword Word Processing Workshop". Contact Interactive.

September 4-6 London: "Using Advanced UNIX Commands". Contact CTG.

September 4-6 London: "UNIX Internals". Contact CTG.

September 9-11 Santa Monica, CA: "UNIX Fundamentals". Contact Interactive.

September 9-12 New York: "UNIX System Administration". Contact LUCID.

September 9-12 New York: "C Programming". Contact USPDI.

September 9-13 Trumbull, CT: "Advanced UNIX". Contact Bunker Ramo.

September 9-13 Chicago and Los Angeles: "C Language Programming". Contact CTG.

September 9-13 Philadelphia: "C Programming Workshop". Contact Plum Hall.

September 9-20 Cincinnati: "UNIX for Application Developers". Contact ITDC.

September 10-12 Trumbull, CT: "Diagnostic UNIX". Contact Bunker Ramo.

September 10-12 Boston and Washington, DC: "UNIX Administration". Contact CTG.

September 10-13 Los Angeles and Washington, DC: "UNIX: A Comprehensive Introduction". Contact ICS.

September 12-13 Santa Monica, CA: "Using the Shell". Contact Interactive.

September 16-17 Santa Monica, CA: "System Administrator's Overview". Contact Interactive.

September 16-17 Chicago and Los Angeles: "Shell Programming". Contact CTG.

September 16-17 Boston and Washington, DC: "Advanced C Programming Workshop". Contact CTG.

September 16-18 Cambridge, MA: "C Technical Seminar". Contact CL Publications.

September 16-19 Callaway Gardens, GA: "UNIX OS: The First Step". Contact AT&T.

September 17-18 Trumbull, CT: "UNIX/C Applications". Contact Bunker Ramo.

September 17-19 St. Louis: "SNA Architecture and Implementation". Contact CSI.

September 17-20 San Diego and Washington, DC: "Programming in C". Contact ICS.

September 17-20 New York: "Advanced C Programming". Contact LUCID.

September 18-20 Chicago and Los Angeles: "Using Advanced UNIX Commands". Contact CTG.

September 18-20 London: "UNIX Administration". Contact CTG.

September 18-20 Boston and Washington, DC: "Advanced C Programming Under UNIX". Contact CTG.

September 18-20 New York: "Comprehensive Overview of the UNIX Operating System". Contact Digital Educational Services.

September 18-20 Santa Monica, CA: "Interactive Networking Tools". Contact Interactive.

September 23-24 London: "Advanced C Programming Workshop". Contact CTG.

September 23-24 Raleigh, NC: "The Concepts of Object Oriented Programming". Contact PPI.

September 23-24 Santa Monica, CA: "Advanced Commands for Programmers". Contact Interactive.

September 23-25 Boston: "C Data Concepts for Programmers". Contact Sessions and Gimpel.

September 23-27 Chicago and Los Angeles: "UNIX Internals". Contact CTG.

September 23-27 Boston and Washington, DC: "Berkeley Fundamentals and `cs` Shell". Contact CTG.

September 23-27 Trumbull, CT: "Advanced C". Contact Bunker Ramo.

September 23-27 Cincinnati: "UNIX Systems Administration". Contact ITDC.

September 24-26 Los Angeles: "SNA Architecture and Implementation". Contact CSI.

September 25-27 London: "Advanced C Programming Under UNIX". Contact CTG.

September 25-27 Santa Monica, CA: "UNIX Architecture—A Conceptual Overview". Contact Interactive.

September 30-October 3 Chicago: "UNIX for Users/UNIX Shell Programming". Contact USPDI.

September 30-October 4 London: "Berkeley Fundamentals and `cs` Shell". Contact CTG.

September 30-October 4 Trumbull, CT: "Intro to UNIX".

Only Sperry can make the following four statements.

Our PC runs the XENIX™ system, as well as MS-DOS™.

Our 4 new microcomputers run the UNIX system.

Our new minicomputer runs the UNIX system.

Our Series 1100 mainframes run the UNIX system.

All of which means there is a great deal we can do for you.

For instance, our family of computers based on UNIX systems has incredible transportability for all your software.

And being able to accommodate from two to hundreds of users, it's impossible to outgrow our hardware.

Of course, this linking of all your computer systems can add measurably to your productivity.

And a fast way to find out

more is to get a copy of our Sperry Information kit. For yours, or to arrange a demonstration at one of our Productivity Centers, call **1-800-547-8362 (ext. 60).**

*UNIX is a trademark of AT&T Bell Laboratories
XENIX and MS-DOS are trademarks of Microsoft Corporation

© Sperry Corporation 1985.



Introducing an idea that makes obsolescence obsolete.



The UNIX* operating system from PC to mainframe.

Circle No. 292 on Inquiry Card

Contact Bunker Ramo.

September 30-October 4 Santa Monica, CA: "The C Programming Language". Contact Interactive.

September 30-October 11 Cincinnati: "C Programming Language". Contact ITDC.

OCTOBER

October 1 New York and Washington, DC: "UNIX Overview". Contact CTG.

October 1-3 Hartford, CT: "SNA Architecture and Implementation". Contact CSI.

October 1-3 Chicago and Los Angeles: "UNIX Administration". Contact CTG.

October 1-4 Baltimore: "UNIX: A Complete Introduction". Contact ICS.

October 1-4 New York: "UNIX System Internals". Contact LUCID.

October 2 Trumbull, CT: "UNIX Marketing". Contact Bunker Ramo.

October 2-4 New York and Washington, DC: "UNIX Fundamentals for Non-Programmers". Contact CTG.

October 3-4 Boston: "C Data Concepts for Managers". Contact Sessions and Gimpel.

October 7-8 Chicago and Los Angeles: "Advanced C Programming Workshop". Contact CTG.

October 7-9 New York and Washington, DC: "UNIX Fundamentals for Programmers". Contact CTG.

October 7-9 Santa Monica, CA: "IS/WB Fundamentals".

Contact Interactive.

October 7-9 New York: "Office Automation". Contact LUCID.

October 7-10 Washington, DC: "C Programming". Contact USPDI.

October 7-11 Absecon, NJ: "C Programming Workshop". Contact Plum Hall.

October 8 London: "UNIX Overview". Contact CTG.

October 8-10 Trumbull, CT: "Diagnostic UNIX". Contact Bunker Ramo.

October 8-11 Baltimore: "Programming in C". Contact ICS.

October 9-11 Chicago and Los Angeles: "Advanced C Programming Under UNIX". Contact CTG.

October 9-11 London: "UNIX Fundamentals for Non-Programmers". Contact CTG.

October 10 Palo Alto, CA: "Introduction to C for Programmers". Contact Berkeley Decision/Systems.

October 10-11 New York and Washington, DC: "Shell as a Command Language". Contact CTG.

October 10-11 Santa Monica, CA: "IS/WB System Administrator's Overview". Contact Interactive.

October 14-15 Santa Monica, CA: "Using Ten/Plus". Contact Interactive.

October 14-15 Columbia, MD: "C Data Concepts for Managers". Contact Sessions and Gimpel.

October 14-16 London: "UNIX Fundamentals for Programmers". Contact CTG.

October 14-18 Cincinnati: "UNIX for End Users". Contact ITDC.

ACUITY® business software is compatible with any budget, and all these systems:

AT&T 3B's	Plexus	Gould
Motorola	Convergent	Sperry
Charles River Data	Cromemco	Momentum
Sun Microsystems	Altos	Dual
All Unix based micros	Harris/VOS	Harris/Unix
All Unix "look-alikes"	VAX/Ultrix	VAX/VMS

Serving general accounting, wholesale, distribution, manufacturing and project/job costing applications on over 30 different machines, ACUITY allows you to select from individual modules to build a fully integrated software system specifically for your needs.

- Accounts Payable • Accounts Receivable
- General Ledger • Fixed Assets • Payroll
- Customer Order Processing • Inventory
- Purchasing/Receiving • Project Management
- MRP • Master Scheduling • BOMP
- Project Scheduling • Labor Projections
- Work Breakdown Structure

For more detailed information, call 619/474-6745.



225 West 30th Street, National City, California 92050

Circle No. 246 on Inquiry Card



Version 3.0 Available Now!

The Reliable High Performance APL for UNIX* Systems

Dyalog APL is fast!

Version 3.0 is up to 10 times faster than previous versions!

Dyalog APL is functional!

- Nested Arrays
- Full Screen Editor
- Full Screen Data Manager
- Event Trapping
- Interface to all UNIX* Facilities
- Optional Graphics

Dyalog APL is reliable!

Dyalog APL has been in commercial use for over two years and is available NOW for most UNIX* Systems so call or write today.

MIPS Software Development, INC.

31555 W. 14 Mile Rd. #104
Farmington Hills, MI 48018
(313) 855-3552

*Improvements are a function of system and usage
*UNIX is a trademark of AT&T Bell Laboratories

Circle No. 245 on Inquiry Card

CONTACT INFORMATION

AT&T Information Systems, Institute for Communications and Information Management, PO Box 8, Pine Mountain, GA 31822-0008. 800/247-1212.

Berkeley Decision/Systems, Inc., 150 Belvedere Terrace, Santa Cruz, CA 95062. 408/458-0500.

Bunker Ramo Information Systems, Trumbull Industrial Park, Trumbull, CT 06609. 203/386-2000.

CL Publications, 131 Townsend Street, San Francisco, CA 94107. 415/957-9353.

Computer Technology Group (CTG), 310 S. Michigan Avenue, Chicago, IL 60604. 800/323-UNIX, or in IL, 312/987-4082.

Communications Solutions, Inc. (CSI), 992 S. Saratoga-Sunnyvale Road, San Jose, CA 95129. 408/725-1568.

Digital Educational Services, Digital Equipment Corp., 12 Crosby Drive, Bedford, MA 01730. 617/276-4949.

Information Technology Development Corp., 9952 Pebbleknoll Drive, Cincinnati, OH 45247. 513/741-8968.

Integrated Computer Systems, PO Box 45405, Los Angeles, CA 90045. 800/421-8166, or in CA, 800/352-8251.

Interactive Systems Corp., 2401 Colorado Avenue, 3rd floor, Santa Monica, CA 90404. 213/453-8649.

LUCID, 260 Fifth Avenue, Suite 901, New York, NY 10001. 212/807-9444.

Plum Hall, 1 Spruce Avenue, Cardiff, NJ 08232. 609/927-3770.

Productivity Products International, Inc. (PPI), 27 Glen Road, Sandy Hook, CT 06482. 203/426-1875.

Sessions & Gimpel Training Associates, 474 Washington Street, Holliston, MA 01746. 617/429-6350.

Silicon Valley Net (SV Net), PO Box 700251, San Jose, CA 95170-0251. 415/594-2821 (Grant Rostig).

Uniq Digital Technologies, 28 S. Water Street, Batavia, IL 60510. 312/879-1008.

US Professional Development Institute (USPDI), UNIX and C Workshops, 1620 Elton Road, Silver Spring, MD 20903. 301/445-4400.

October 14-18 Absecon, NJ: "Advanced C Topics Seminar". Contact Plum Hall.

October 15 Dallas and San Francisco: "UNIX Overview". Contact CTG.

October 15-17 New York: "C Language Fundamentals". Contact LUCID.

October 15-18 Palo Alto, CA: "UNIX: A Complete Introduction". Contact ICS.

October 16-18 Dallas and San Francisco: "UNIX Fundamentals for Non-Programmers". Contact CTG.

October 16-18 Santa Monica, CA: "Customizing Ten/Plus". Contact Interactive.

October 16-18 Columbia, MD: "C Data Concepts for Programmers". Contact Sessions and Gimpel.

October 17-18 London: "Shell as a Command Language". Contact CTG.

October 21-22 Boston: "The Concepts of Object-Oriented Programming". Contact PPI.

Please send announcements about training or events of interest to: UNIX Review Calendar, 500 Howard Street, San Francisco, CA 94105. Include the sponsor, date and location of event, address of contact, and relevant background information.

68000 68010 **68020** SOFTWARE TOOLS

**WE ARE PROUD TO ANNOUNCE THE BIRTH OF
THE NEWEST MEMBERS OF OUR 68000 FAMILY
... YOUR 68020 TOOLS ARE HERE!**

TOOL KIT

- 68000/10/20 Assembler Package:
 - Macro Cross/Native Assembler
 - Linker and Librarian
 - Cross Reference Facility
 - Symbol Formatter Utility
 - Object Module Translator
- Green Hills C 68000/10/20 Optimizing Compilers
- Symbolic Debuggers

AVAILABILITY

VAX, microVAX, 8600, Sun, Pyramid, Masscomp, IBM/PC, OASYS Attached Processors for VAX and PC, others. Runs under VMS, Bsd 4.2, System V, MS/DOS, dozens more.

Your name it . . .

We provide a "One-Stop Shopping" service for more than 100 products running on, and/or targeting to, the most popular 32-, 16- and 8-bit micros and operating systems.

FEATURES

- Written in C; fast, accurate, portable.
- Supports 68000 and 68010.
- 5,000 line test suite included.
- EXORmacs compatible.
- Produces full listings and maps.
- Outputs S-records and Tek-Hex formats.
- Runs native or cross. Extensive libraries.
- Supports OASYS compilers.
- Generates PROMable output and PIC.
- Full Floating Point support.

Over 100 Other OASYS software tools to choose from.

A DIVISION OF XEL

OASYS

60 Aberdeen Avenue, Cambridge, MA 02138 (617) 491-4180

Circle No. 244 on Inquiry Card

THE LAST WORD

Letters to the Editor

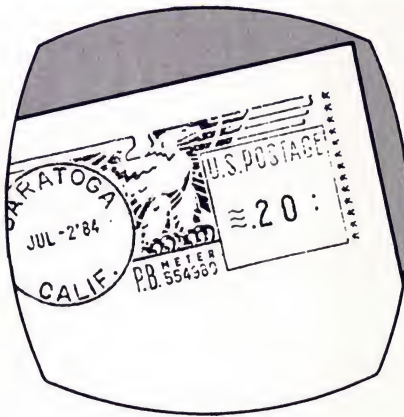
THE ZERO CAPER

Dear UNIX REVIEW,

One of the things I hate about editing a magazine, as I used to do, is chasing typographicals. Writing notes about them is truly painful.

But there is a typo in the February edition in which the Zilog Systems Series 2 product release appears [Recent Releases], that may cause tangible problems, so I thought I'd better note it.

The story indicates that the machine offers "support for up to four users." In fact, the new Series 2 supports 40 users. The difference could be material; some of the sales guys are complaining that it makes them look like a Fortune system.



Dick Davies
Zilog, Inc.
San Francisco

BUGS? WHERE?

Dear UNIX REVIEW,

Please check things a bit more closely before suggesting they're bugs! On page 77 of the April issue [Problem Solver], you wrote: "In system V, the bug can be fixed . . ."

It is not necessary to fix the System V **init** to do what you want. In fact, you're doing something I see quite often. You're "fixing" a new version to look more like an old version rather than bother to learn any new features! System V **init(1M)/inittab(4)** provides an extremely powerful tool for automatic configuration under virtually any situation. I wish you'd spend some time trying to understand how it works. If there is a change in a new release of UNIX, it is clearly there for a purpose!

Just some facts about System V (you can look up

the details):

1) Via */etc/inittab*, it is trivial to invoke a shell that goes through root's (or another user's) *.profile* by executing as the command **su - [username]**. Hence, the "bug fix" is not necessary. (The "-" flag says **fork** a **-sh** shell.)

2) Further, it's not necessary to put your administrative stuff in *.profile* (nor is it recommended). The */etc/rc* file exists for this purpose. Your "30 seconds to

stop me before going multi" is in fact a matter of structuring your *rc* file thus:

```
# get current and previous run level
set=who -r
RunLevel=$7
LastRL=$9
case "$RunLevel" in
  1) # Single User
    echo "Going Multi in 30 seconds"
    echo "(type DEL/Break to interrupt)"
    sleep 30
    init 2
    ;;
  2) # Level 2 (Multiuser)
    ...
    whatever actions needed
    ...
    ;;
esac
```

Since **who -r** also provides info on the "previous run level" and the number of times the current run level has been entered since boot, the *rc* file can handle conditions such as "the first time entering multiuser after boot" and "entering level N after

UNIX EXPO™



For one week in September

New York City becomes the heart of the UNIX universe

Join the thousands of your colleagues who will seek answers to meet their business needs... and come away with a full understanding of UNIX solutions and applications.

Take advantage of the best UNIX has to offer:

- An exhibition featuring over 200 of the leading suppliers of UNIX based hardware, software and services.

- A tutorial program designed and developed by AT&T — the most respected source for the UNIX System.
- A conference examining the advantages of UNIX solutions in the business environment.

Plan now to attend and profit from THE PROVEN UNIX MARKETPLACE.

For all the details contact: UNIX EXPO 14 West 40th Street, New York, N.Y. 10018 Telephone: 212-391-9111. TELEX: 135401 DIMCOMM.

Circle No. 299 on Inquiry Card

UNIX™ is a registered trademark of Bell Labs. UNIX EXPO is not affiliated with Bell Labs.

being at level M". Further, **inittab** includes action designators such as **boot** and **powerfail** that allow additional shell scripts or commands to be invoked only under these special circumstances.

3) The test "if (\$\$ < 5) . . ." is, of course, a pretty sloppy way to determine if **.cshrc** was invoked by **init**. Depending on which version of UNIX you're using, there may be more or less processes run before the execution of **csh**. If more, \$\$ => 5; if less, \$\$ may in fact be reassigned when the process IDs wrap around to 1 again. (Typically, the maximum pid is 30000. Then pid numbers start over, skipping any pids still in use.) Of course, with the System V features listed above, this test is not necessary.

4) By using **.cshrc**, you're encouraging what may be a real performance hog. Note that **.cshrc** is invoked *every time* a new **csh** is forked. This can lead to a lot of unnecessary command execution that the user typically does not know about. In your example, whenever user root happens to execute a **csh** script, the "if (\$\$ < 5..." code is executed again. (Also think about what happens if you have

SHELL=csh and do a **make!**)

5) Just about everything you mentioned in the way of automation has already been made available by a number of vendors, including AT&T on the 3B2. We, in the UNIX community, need to get beyond telling our users how to do things like you've described in your article. UNIX will never be accepted in the business world if every reader of UNIX REVIEW starts hacking their **inittabs**. It's very powerful, and lots of fun, but let's package all this stuff so the users can get their job done and we can move on to more productive hacking.

Finally, I'd recommend you get **ksh** (the Korn shell) from the AT&T "Toolchest". Now that **ksh** is available outside of AT&T (and the code compiles and runs on everything from PC/IX to System V, 4.2BSD, V8, and maxi-UNIX), there's simply no reason for anyone to consider using **csh**. (Unless, of course, it's "loyalty to the past". I know a number of people that still use **ed** instead of **vi**.)

Chuck Flink
AT&T Technologies
Fredericksburg, VA

New from Image Network!

Documenter's Workbench[®]

for laserprinters and typesetters.

DWB is *troff*, *eqn*, *tbl*, and *pic* interfaced to raster printing devices.

Our existing **XROFF** product allows **DWB** to work with the following systems and printers:

- | | |
|---------------------|--------------------|
| • System V | • System III |
| • Berkeley 4.2 | • V 7 |
| • VAX/Ultrix | • VAX/VMS |
| • IBM/PC MS/DOS | • Amdahl/UTS |
| • Eunice | • Xenix |
| • UniPlus+ | • UNOS |
| • DEC LN01s, LN03 | • Xerox 2700, 3700 |
| • APS-5 typesetter | • Xerox 8700, 9700 |
| • Compugraphic 8400 | |

Use **DWB** with a laser printer to make high quality documents or to make proof copies before typesetting.

Call or write to tell us *your* printing requirements!

Image Network, (408)746-3754,
424 Palmetto Drive, Sunnyvale, CA, 94086-6760

*Documenter's Workbench is a trademark of AT&T Bell Laboratories.

Q-CALC

A superior spreadsheet on UNIX*

As powerful as Lotus 1-2-3*

- large spreadsheet
- many business functions
- complete GRAPHICS package
- translates 1-2-3 models into Q-CALC
- already ported to: VAX, Callan, Fortune, 3B2, Cyb, Plexus, Codata, Cadmus, Masscomp, Sun, etc.
- Ideal for VARs/ISVs

Available since Jan. '84
For more information write/call
Quality software Products
348 S. Clark Drive
Beverly Hills, CA 90211
213-659-1560

*Lotus 1-2-3 is a trademark of Lotus Development Corp. UNIX is a trademark of AT&T.

Circle No. 249 on Inquiry Card

Circle No. 250 on Inquiry Card

COMPLETE YOUR UNIX REVIEW LIBRARY!



- June/July 1983—UNIX on the IBM/PC
- August/September 1983—Sritek and Venix . .
- October/November 1983—UNIX Typesetting
- December/January 1984—Vi and Emacs . . .
- February/March 1984—UNIX Databases . . .
- April/May 1984—Menu-based User Interfaces
- June 1984—Big Blue UNIX
- July 1984—The AT&T Family
- August 1984—Documentation
- September 1984—System Administration . . .
- October 1984—UNIX on Big Iron
- November 1984—User Friendly UNIX
- December 1984—Low Cost UNIX
- January 1985—Evolution of UNIX
- February 1985—UNIX Portability
- March 1985—Performance
- April 1985—UNIX Networking
- May 1985—Distributed Resource Sharing . . .
- June 1985—UNIX Applications
- July 1985—Office Automation
- August 1985—Database Intricacies
- September 1985—Languages

Back issues are \$4.95 each including postage. Payment in advance is required. Send this order form with check (US funds payable at US bank only) or credit card information to: REVIEW Publications, 901 S. 3rd St., Renton, WA 98055. Additional \$1.00/issue for foreign mail.

Name _____
 Company _____
 Address _____
 City _____ State ____ Zip _____
 M/C or VISA # _____
 Exp. Date _____

ADVERTISER'S INDEX

Absoft	94	Interactive Systems Corp.	13
Access Methods Inc.	32	Lifeboat Associates	93
Adax Inc.	90	Lions Gate Software	89
Advanced Ergonomic Management	86	Marc Software	74,75
AT&T Information Systems	67	Micro Communications	95
B.A.S.I.S.	76	MIPS Software	102
Basmark Corp.	62	NETI	Centerspread
bbj Computer Services	75	N.I.A.L.	98
Bell Technologies	11	Oasys	103
Brandon Consulting	49	Prior Data Sciences	79
Ceegen Corp.	91	Quality Software Products	106
Century Software	87	Raima Corp.	60
Cleo Software	71	Relational Database Systems	1,2,3
CMI Corp.	33	Robert Half	86
Cogitate	86	Santa Cruz Operation	47
Communications Research	32	Scientific Placement	66
Computer Cognition	102	Software Development Systems	41
Computer Technology Group	73	Silicon Valley Software	25,27,29,31
Concentric Associates	97	Sperry Corp.	101
Corporate Microsystems	19	Syntactics	15
COSI	83	Ubiquitous Systems	96
DCC Data Service	98	Unbound, Inc.	26
DSD Corp.	17	Unipress Software	59,61,63,65
Dynacomp	77	Unitech	16
Emerging Technology Inc.	9	University of Toronto	81
Franz, Inc.	21	UNIX Expo	105
Gould	34	UX Software	Cover IV
Greenhills Software	58	Verdix	30
Handle Technologies	Cover II	XED/Computer Methods	7
Image Network	106	Zanthe Information Systems	Cover III
Information Technology Development Corp.	57		

COMING UP IN OCTOBER

UNIX and Universities

- **The Symbiotic Relationship**
- **Historical Perspectives**
- **Keeping UNIX Fresh**
- **Benefits to Industry**
- **Dangerous Blind Spots**

"Nature Imposes Few Restrictions on Those Daring Enough to Lead"

ZIM is a fully integrated fourth generation application development system designed for leading system integrators and corporate and independent applications developers.

*COMPLETE DEVELOPMENT ENVIRONMENT

- Report Writer
- Forms Painter and Manager
- Data Dictionary
- Application Generator
- Non-procedural Programming Language
- Compiler
- C Language Interface
- Runtime System

*POST RELATIONAL

- Entity Relationship Model
- Powerful extension of Relational Model

*MAINFRAME POWER, FUNCTIONALITY AND PERFORMANCE

*APPLICATIONS PORTABILITY

- MS-DOS, UNIX, XENIX, and QNX

*MULTI-USER

- Full transaction processing control

*NETWORKING

*APPLICATIONS LIMITED ONLY BY HARDWARE

*BUILT-IN STRATEGY OPTIMIZER

*ENGLISH-LIKE LANGUAGE
*QUALITY PRODUCT SUPPORT
ZIM is a mainframe system that runs on micro-computers and on super micro-computers. If you want mainframe power, speed, flexibility and freedom from arbitrary limitations all at a micro price, talk to us about an evaluation system.
Dealer inquiries are welcome.



1785 Woodward Dr., Ottawa, Ontario
K2C 0R1 (613)727-1397

MS-DOS and XENIX are Microsoft Corp. trademarks.
UNIX is an AT&T trademark. QNX is a Quantum Software Systems trademark.



Now available for XENIX
on the IBM PC AT.



The Language for a New Generation

Portability. UX-Basic™ application programs execute unchanged on any UNIX™ machine and are completely device independent.

Power. UX-Basic contains the building blocks for efficient application program development. It also allows you to tap the full power of UNIX and gives you direct access to data bases.

Productivity. UX-Basic is friendly and easy to learn and use. The interactive programming environment provides syntax checking as well as real-time debugging and testing.

Performance. UX-Basic gives you speed when you need it with our efficient pseudo-code compiler/runtime package. We are constantly working to keep UX-Basic's performance at the leading edge.

Profit. UX-Basic programs are structured, modular and readable. Maintenance and support are easy.

Perfect for UNIX . . . a new generation of computers . . . a new generation of computer users.

UX Software, Inc.

10 St. Mary Street, Toronto, Canada M4Y 1P9
Tel: (416) 964-6909 TLX: 065-24099

Available from major computer manufacturers such as Altos, AT&T, Siemens and an international network of distributors.

See us at Booth #246 at UNIX Expo in New York City.

UNIX is a Trademark of AT&T laboratories
UX-Basic is a Trademark of UX Software, Inc.

See us at
COMDEX/Fall '85
November 20-24, 1985
Las Vegas Convention Center-West Hall
Las Vegas, Nevada