COMPUTER LANGUAGE™

$2.95
Canada $3.95

VOLUME 2, NUMBER 7

JULY 1985

SPEAKING LISP

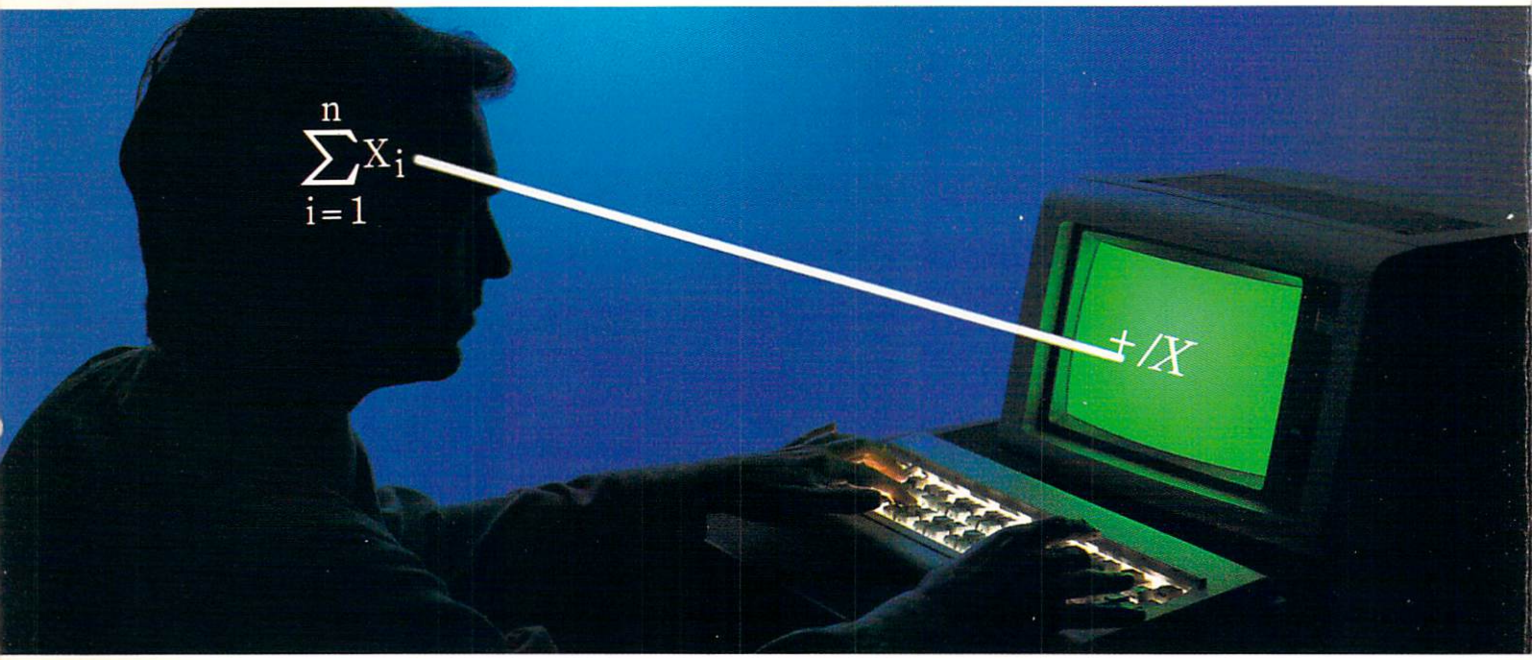
PRODUCT WRAP-UPS: EXPERT SYSTEMS, LISP, PROLOG

AI PIONEER TERRY WINOGRAD

INTERACTIVE DECISION TABLE SOFTWARE

SPECIAL ISSUE: ARTIFICIAL INTELLIGENCE

SOLVE PROGRAMMING PROBLEMS THE WAY YOU THINK. PURE AND SYMBOL.



APL★PLUS®/PC IS THE ANSWER.

The shortest distance between two points is a straight line. But unfortunately, that's not the case in programming.

Most languages require you to go through an enormous number of steps before an idea becomes reality.

That's why the APL★PLUS/PC System is such a dramatic and exciting software tool for serious PC programmers and application developers.

Instead of requiring you to learn—and write—long-winded and complicated programs, APL is based on your instinctive ability to deal in symbols. And once you begin using APL's quick notations, you'll find it the ideal programming environment for all your application needs.

The incredible shortcuts you'll get with APL not only make you more productive, but make programming enjoyable. Intricate calculations and modeling on PC's are a snap. You'll spend less time on drudgery, and more time creating.

Only with APL★PLUS/PC, do you get:
- full-screen editing
- a built-in terminal emulator
- communications
- graphics primitives
- and report formatting.

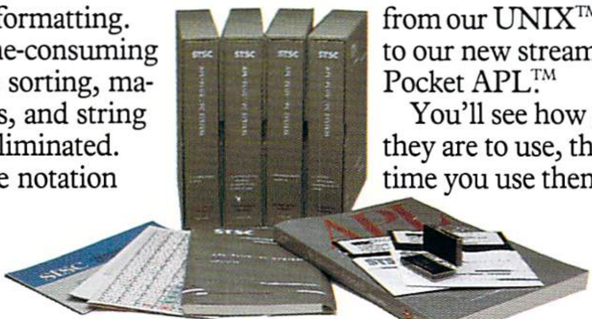
Writing time-consuming programs like sorting, matrix inversions, and string searching is eliminated. APL's concise notation already provides these. . .and more.

No wonder a *PC Magazine* reviewer enthusiastically reacted to our APL★PLUS/PC System with "awe and delight."

So will you. The complete package price is $595 and major credit cards are accepted.

Act now and we'll send you a free Convincer Kit. Contact your local dealer, or call **800-592-0050** (in Maryland, call **301-984-5123**) to order your system, or for more information about our other APL PLUS★WARE™ products—from our UNIX™ version to our new streamlined Pocket APL.™

You'll see how symbol they are to use, the very first time you use them.



Problem-solving at the speed of thought.

STSC
A Contel Company

APL★PLUS/PC System requires 192K. A soft character set can be used for computers with IBM compatible graphics board. A character generator ROM or software is included for the IBM PC or selected compatibles.

PLUS★WARE and POCKET APL are trademarks of STSC, Inc. APL★PLUS is a registered service mark and trademark of STSC, Inc. UNIX is a trademark of AT&T Bell Laboratories.

CIRCLE 121 ON READER SERVICE CARD

Software development isn't a mountainous task once you eliminate the high C errors.

When you can find and fix bugs at the earliest possible moment, creating software stops being such an uphill grind.

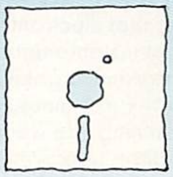
And the Smart/C Environment makes it possible. It's a complete, fully-integrated development environment for C that saves you from the creativity-inhibiting cycle of edit, compile, re-edit, re-compile, link, load, test, re-edit, re-compile, etc., ad infinitum. Smart/C puts the fun back in programming, because you spend your time creating... not waiting.

Here's why. Syntax errors are eliminated automatically as code is entered. Smart/C's highly integrated editor and interpreter allow you to interpret your program at any time in the creation process, so logic errors can be ferreted out as soon as the algorithm exists—long before any compile, link, or load.

The complete integration of the editor and interpreter means you can stop anywhere in the interpret cycle, edit, and then go right back into the interpreter exactly where you left off. Not only that, the screen-oriented user interface lets you see all operations, even interpretation, right on the listing of the code.

And to make maintenance programming easier, Smart/C's Migrator allows existing C code produced with any editor to be modified and run within the Smart/C Environment.

All of which makes Smart/C an excellent tool. It's flexible, non-restrictive, and lets you create elegant, readable, error-free programs that you can watch run with a great feeling of satisfaction.



Smart/C™

Free Demo Disk!

To fully appreciate Smart/C, you have to see it in action. For your free IBM PC MS-DOS demo disk, call us. Or write us on your company letterhead.
AGS Computers, Inc., Advanced Products Division, 1139 Spruce Drive, Mountainside, NJ 07092.
800-AGS-1313. In NJ, 201-654-4321.

Smart/C Features

The Smart/C Environment
- ☐ Fully integrated editor and interpreter
- ☐ Only one load brings them both in
- ☐ One command set
- ☐ Move between one another at will

Syntax Directed Editor
- ☐ vi-like command set
- ☐ Automatically provides formats for blocks, *for, case* and *if* statements

Interpreter
- ☐ Current module can call external modules during interpretation
- ☐ Has Include capability
- ☐ Totally precompilation—no incremental compile
- ☐ Can interpret partially defined files allowing for rapid prototyping
- ☐ Variable speed of interpretation
- ☐ Multiple windows with user-defined sizes

The Smart/C Migrator
- ☐ Allows C code produced with any editor to be interpreted by Smart/C
- ☐ Reformats for readability

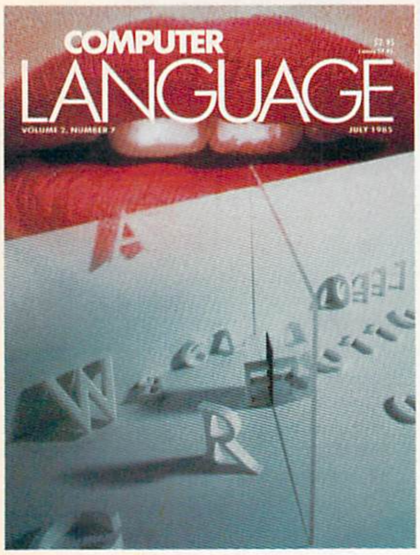
Smart/C has been ported to UNIX™ System V Release 2, Berkeley 4.2, Xenix,™ and MS-DOS. Versions run on 8086- and 68000-based machines, as well as proprietary architectures. Smart/C runs on PCs, micros, supermicros, minis, and even mainframes.

AGS

CIRCLE 7 ON READER SERVICE CARD

EDIT
COMPILE
TEST RE-EDIT
RE-COMPILE RE-TEST
RE-EDIT RE-COMPILE RE-
TEST RE-EDIT RE-COMPILE RE-
TEST RE-EDIT RE-COMPILE RE-TEST
RE-EDIT RE-COMPILE RE-TEST RE-EDIT
RE-COMPILE RE-TEST RE-EDIT RE-COMPILE
E-TEST RE-EDIT RE-COMPILE RE-TEST RE-EDIT
-COMPILE RE-TEST RE-EDIT RE-COMPILE RE-TES
T RE-COMPILE RE-TEST RE-EDIT RE-COMPILE RE

COMPUTER LANGUAGE



ARTICLES

Speaking Lisp 27
by John R. Allen
Traditional languages, like FORTRAN or Pascal, are much less confusing to the human reader . . . or so the claim goes. In this article, we explore the deeper semantic issues that separate LISP from the programming language mainstream. By studying a specific example—how to compute a sum given a finite sequence of numbers—we can observe LISP's powerful style of computational notation.

Logic at a Glance, Part I 35
by Jim McCarthy
An electronic decision table provides many of the same qualitative experiences to the user that electronic spreadsheets do: a visceral sense of control, the feeling of instantaneity, rapid-fire accuracy, and the joy of liberating latent programming skills. In this first installment of a three-part series, Jim McCarthy examines the structure and value of basic decision table programming. He will contribute the complete source code to a prototype editor, interpreter, and compiler.

Natural Language Software 43
by Darryl Rubin
If you have ever written a compiler, you will recognize immediately that natural language processing can be divided into three phases: lexical, syntactic, and semantic analysis. In this article, we will study the particular problems posed by implementing this new technology on microcomputers and look at two natural language data base programs on the market which try to tackle these problems.

Learning about PROLOG 49
by Ramachandran Bharath and Margaret Sklar
A PROLOG program consists of declarations of relationships between objects. During program execution, PROLOG draws logical deductions from facts or relationships the user has supplied. The purpose of this article is to give an indication, through simple examples in PROLOG, of what is distinctive about the PROLOG programming style and to look at why it is especially suitable for user-friendly applications.

DEPARTMENTS

Editor's Notes 4
Feedback 7
CrossThoughts 11
B+ trees, B++ trees, and statistics in AI
ComputerVisions 19
AI innovator Terry Winograd
Public Domain Software Review 57
Screen utilities and a gnu
Exotic Language of the Month Club 61
MRS: An experimental AI system
Product BINGO 69
Software Review 71
Knowledge systems for the IBM PC, Part I
Software Review 85
Shopping for a LISP
Software Review 95
PC PROLOGs
Advertiser Index 112

New Version

Mighty Macro Assembler.

The new Microsoft® Macro Assembler package. A complete development environment that makes you a more productive programmer. Whether you're using Macro Assembler or any Microsoft high level language.

A common calling convention lets you easily call assembly language routines from any high level Microsoft language to add an extra burst of blinding speed.

Better Debugging.

The new Symbolic Debug Utility lets you stay close to the source. Now you can step through your assembled or compiled code by name rather than by address. Source level display for Microsoft Pascal, FORTRAN, and C allows you to view both your original source and the resulting code.

And we stuffed our package with a full set of the most useful utilities around. So that you can link, maintain and organize your programs like never before.

Who else but Microsoft could build so much into one package for $150?

For the name of your nearest Microsoft dealer call (800) 426-9400. In Washington State, Alaska, Hawaii and Canada, call (206) 828-8088. And if you're already using Microsoft or IBM® Macro Assembler, ask us how you can upgrade to the mightiest Macro of them all.

MICROSOFT®
The High Performance Software™

Microsoft Macro Assembler Package:

Macro Assembler
- For the 8086/8087/8088 and now the 186/286/287.
- Define macros.
- Conditional Assembly.
- Case sensitivity for symbols.

New Interactive Symbolic Debug Utility
- Controlled testing environment for debugging.
- Source line display of Microsoft FORTRAN, Pascal and C Programs.
- Set breakpoints on line numbers and symbols.
- Single step to follow program execution.
- Disassemble object code.
- Display values.
- Make minor changes without reassembling.

New Program Maintenance Utility
- Rebuilds your applications after your source files have been changed.
- Similar to UNIX™ Make utility.

Library Manager
- Create, organize and maintain your object module libraries created with Microsoft Languages.
- Set page size (default of 16 bytes).

Object Code Linker
- Simple overlaying linker combines relocatable object modules created using Microsoft Languages into a single program.
- Load Map generation.
- Specify from 1 to 1024 segments.

Cross Reference Utility for the Macro Assembler
- Creates a cross-reference listing of the definitions and locations of all symbols used in an assembly language program.

COMPUTER LANGUAGE

EDITOR
Craig LaGrow

MANAGING EDITOR
Regina Starr Ridley

TECHNICAL EDITOR
John Halamka

PRODUCT REVIEW EDITOR
Hugh Byrne

EDITORIAL ASSISTANT
Kathy Kincade

CONTRIBUTING EDITORS
Nick Flann, Doug Millison, Tim Parker, John Sechrest, Namir Clement Shammas

SPECIAL PROJECTS MANAGER
Jan Dente

OPERATIONS CONSULTANT
Beatrice C. Blatteis

CIRCULATION COORDINATOR
Renato Sunico

ART DIRECTOR
Jeanne Schacht

COVER PHOTO
Dow/Clement Photography

PRODUCTION ARTISTS
Anne Doering, Barbara Luck

PRODUCTION
Steve Campbell, Kyle Houbolt, Julie Oxberry

TECHNICAL CONSULTANT
Addison Sims

ACCOUNTING MANAGER
Lauren Kalkstein

WHOLESALE COORDINATOR
Nicola Sullivan

PUBLISHER
Carl Landau

COMPUTER LANGUAGE *(ISSN 0749-2839) is published monthly by COMPUTER LANGUAGE Publishing Ltd., 131 Townsend St., San Francisco, CA 94107. (415) 957-9353.*

Advertising: For information on ad rates, deadlines, and placement, contact Carl Landau at (415) 957-9353, or write to: COMPUTER LANGUAGE, 131 Townsend St., San Francisco, CA 94107.

Editorial: Please address all letters and inquiries to: Craig LaGrow, Editor, COMPUTER LANGUAGE, 131 Townsend St., San Francisco, CA 94107.

Subscriptions: Contact COMPUTER LANGUAGE, Subscriptions Dept., 2443 Fillmore St., Suite 346, San Francisco, CA 94115. Single copy price: $2.95. Subscription prices: $24.95 per year (U.S.); $30.95 per year (Canada and Mexico). Subscription prices for outside the U.S., Canada, and Mexico: $36.95 (surface mail), $54.95 (air mail)—U.S. currency only. Please allow six weeks for new subscription service to begin.

Postal information: Second-class postage is paid at San Francisco, CA and at additional mailing offices.

Reprints: Copyright 1985 by COMPUTER LANGUAGE Publishing Ltd. All rights reserved. Reproduction of material appearing in COMPUTER LANGUAGE is forbidden without written permission.

Change of address: Please allow six weeks for change of address to take effect. POSTMASTER: Send change of address (Form 3579) to COMPUTER LANGUAGE, 131 Townsend St., San Francisco, CA 94107.

COMPUTER LANGUAGE is a registered trademark owned by the magazine's parent company, CL Publications. All material published in COMPUTER LANGUAGE is copyrighted © 1985 by CL Publications, Inc. All rights reserved.

Editor's Notes

Great news! I have a special announcement that will interest all those readers who have written in stating they think *COMPUTER LANGUAGE* should print alongside its articles any listings referred to in the magazine, regardless of their length. This announcement is for those people who either do not have modems or who are not interested in making the long distance telephone call to our remote bulletin board computers to download the code.

We've started the *COMPUTER LANGUAGE* Users Group . . .

From now on, and for all the issues we've published in the past, all the code referred to in the magazine can be acquired in any disk format by sending $6.50 for every 5¼-in. disk and $8.50 for every SSSD 8-in. disk to: *COMPUTER LANGUAGE* Users Group, 131 Townsend St., San Francisco, Calif. 94107.

Just specify which disk format you would like the code converted onto, and depending on the amount of space your disk will hold, determine how many disks you will need to purchase. Include the name and address where you would like the disks to be mailed. Each month, on this page of the magazine, you'll find a reference to how many kilobytes of source code the current issue's code consumes. The following table should be used as a guide:

Volume	Issue	Size
1	Premier '84	130K
2	Oct. '84	42K
3	Nov. '84	150K
4	Dec. '84	140K
5	Jan. '85	40K
6	Feb. '85	190K
7	Mar. '85	230K
8	Apr. '85	230K
9	May '85	150K
10	June '85	80K
11	July '85 (not incl. ERGO Logic Kit code)	100K
12	July '85 (ERGO Logic Kit code only)	500K
13	All Code Swap Shops	350K

We see the *COMPUTER LANGUAGE* Users Group as an ideal solution to solving many problems at the same time: it helps us free up more pages in the magazine to publish a greater number of informative articles, it allows readers to pick and choose the listings they need, and it allows us to provide readers with machine-readable listings on all floppy disk formats so they can be used immediately without the labor of rekeying. Having listings available on disks replaces our previous policy of mailing out printed listings.

In addition to our comprehensive product analyses of Expert Systems, LISPs, and PROLOGs, this month features a special comparative review of the syntax and semantics of the LISP language vs. other mainstream languages, a massive source code contribution by Jim McCarthy on decision table software, an overview of the field of natural language software, an interview with AI innovator Terry Winograd, and a lot more . . .

Many readers have expressed a desire for *COMPUTER LANGUAGE* to cover the AI field in greater intensity. Some have written in asking us to devote a column to artificial intelligence programming techniques and trends. What do you think?

Your comments and criticisms have been extremely helpful to us in this, our first year of publication. Our warmest thanks!

Craig LaGrow
Craig LaGrow
Editor

Telecommunicate to COMPUTER LANGUAGE

COMPUTER LANGUAGE has established two bulletin board systems for you to upload and download text and binary programs, as well as to leave your own electronic Letter to the Editor. All the program listings referred to in every issue of the magazine will be available here.

In addition, *COMPUTER LANGUAGE* has its own Special Interest Group on CompuServe's national data base. After calling into your local CompuServe node, simply type "GO CLM" at any prompt and you'll be in our SIG.

To access our bulletin board, set your computer or terminal to the following parameters: 8 data bits, no parity, 1 stop bit, full duplex, and either 300 or 1200 baud. The telephone number is (415) 957-9370. After your modem makes the connection, type RETURN several times, and everything else is easy.

Both systems are open 24 hours per day, 7 days per week. Due to the heavy number of callers, please do not log into the system more than one time per day. Messages left on either system will be combined the following day.

Potent Pascal.

Microsoft® Pascal may be the most powerful software development environment available for the MS™-DOS system. It combines the programming advantages of a structured high-level language with the fast execution speed of native code compilation.

And it exceeds the proposed ISO and ANSI standards with logical extensions that make the language more powerful and versatile. For example, programming capabilities even allow you to manipulate data at the system and machine level.

It gives you single and double precision IEEE floating point arithmetic. Numeric operations take advantage of the 8087. Or automatic software emulation is provided if the coprocessor is not installed.

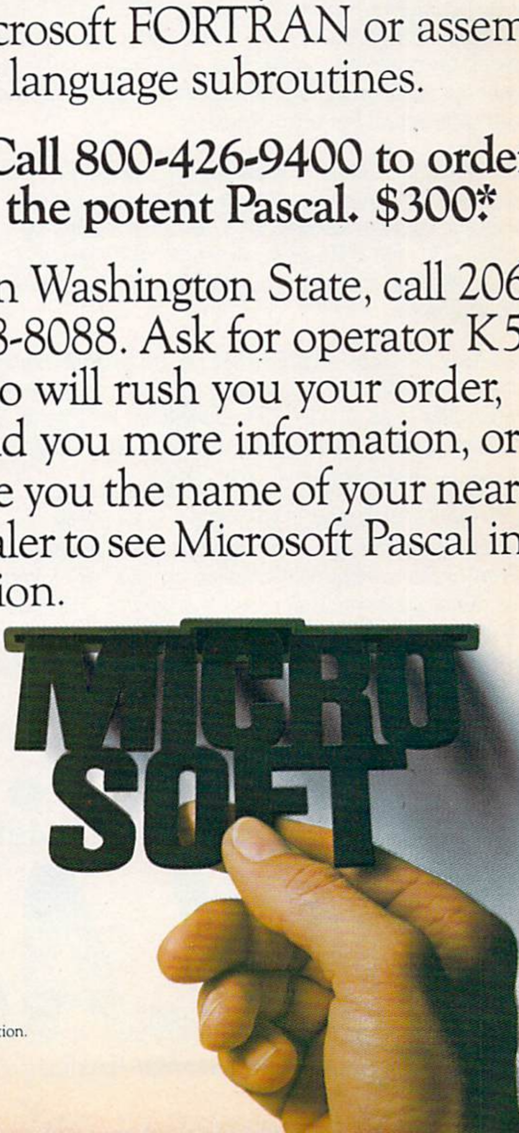
MICROSOFT.
The High Performance Software

Support for long heap allocation and separate module compilation gives you the flexibility to create large programs up to one megabyte.

And the standard linking interface makes it easy to combine Microsoft FORTRAN or assembly language subroutines.

Call 800-426-9400 to order the potent Pascal. $300.*

In Washington State, call 206-828-8088. Ask for operator K5, who will rush you your order, send you more information, or give you the name of your nearest dealer to see Microsoft Pascal in action.



*Price exclusive of handling and Washington State sales tax.

PERFORMANCE PACKAGE

Blaise Computing Inc. introduces the PERFORMANCE PACKAGE™ for Turbo Pascal programmers.

TURBO PASCAL

Turbo ASYNCH™

With Turbo ASYNCH, you can be in constant touch with the world without ever leaving the console. Rapid transit at its best. Turbo ASYNCH is designed to let you incorporate asynchronous communication capabilities into your Turbo Pascal application programs, and it will drive any asynchronous device via the RS232 ports, like printers, plotters, modems or even other computers. Turbo ASYNCH is fast, accurate and lives up to its specs. Features include...

◆ Initialization of the COM ports allowing you to set all transmission options. ◆ Interrupt processing. ◆ Data transfer between circular queues and communications ports. ◆ Simultaneous buffered input and output to both COM ports. ◆ Transmission speeds up to 9600 Baud. ◆ Input and output queues as large as you wish. ◆ XON/XOFF protocol.

The underlying functions of Turbo ASYNCH are carefully crafted in assembler for efficiency, and drive the UART and programmable interrupt controller chips directly. These functions, installed as a runtime resident system, require just 3.2K bytes. The interface to the assembler routines is written in Turbo Pascal.

NEW!

Turbo POWER TOOLS™

Turbo POWER TOOLS is a sleek new series of procedures designed specifically to complement Turbo Pascal on IBM and compatible computers. Every component in Turbo POWER TOOLS is precision engineered to give you fluid and responsive handling, with all the options you need packed into its clean lines. High performance and full instrumentation, including...

◆ Extensive string handling to complement the powerful Turbo Pascal functions. ◆ Screen support and window management, giving you fast direct access to the screen without using BIOS calls. ◆ Access to BIOS and DOS services, including DOS 3.0 and the IBM AT. ◆ Full program control by allowing you to execute any other program from within your Turbo Pascal application. ◆ Interrupt service routines written entirely in Turbo Pascal. Assembly code is not required even to service hardware interrupts like the keyboard or clock.

Using Turbo POWER TOOLS, you can now "filter" the keyboard or even DOS, and create your own "sidekickable" applications.

The Turbo Pascal PERFORMANCE PACKAGE™ is for the serious Turbo Pascal programmer who wants quality tools to develop applications. Every system comes with a comprehensive User Reference Manual, all source code and useful sample programs. They require an IBM PC or compatible, utilizing MS-DOS version 2.0 or later. There are no royalties for incorporating PERFORMANCE PACKAGE functions into your applications.

Turbo POWER TOOLS and Turbo ASYNCH sell for $99.95 each, and they may be ordered directly from Blaise Computing Inc. To order, call (415) 540-5441.

YES, send me the Best for the Best! Enclosed is ________ for
☐ Turbo ASYNCH ☐ Turbo POWER TOOLS. (CA residents add 6½% Sales Tax. All orders add $6.00 for shipping.)

Name: ________ Phone: ________
Shipping Address: ________
City: ________ State: ________ Zip: ________
VISA or MC #: ________ Exp. Date: ________

Turbo Pascal is a trademark of Borland International. Turbo POWER TOOLS, Turbo ASYNCH and PERFORMANCE PACKAGE are trademarks of Blaise Computing Inc. IBM is a registered trademark of Internationl Business Machines Corporation. MS-DOS is a trademark of Microsoft Corporation.

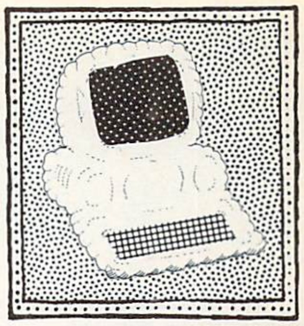
BLAISE COMPUTING INC.

BLAISE watch us!

◆ 2034 BLAKE STREET
◆ BERKELEY, CA 94704
◆ (415) 540-5441

CIRCLE 3 ON READER SERVICE CARD

FEEDBACK



BASIC recursion

Dear Editor:

I am writing to comment on Hugh Aguilar's "BASIC Recursive Techniques" (May 1984, pp. 43-46). I feel that several points need attention.

■ Figure 1 is referred to in the text as an English algorithm for the puzzle's solution, but the figure printed is a picture of the disks and pegs. The cited algorithm was left out entirely.

■ I don't know what Listing 2 is supposed to be—a replacement for the A.1 and A.3 sections of the missing Figure 1 perhaps?— but it is hard to decipher with the = and + symbols not printed. (Alas, why won't typewriter makers understand programmers so we won't have to keep remembering to draw in the =, +, <, >, etc.?)

By the way, the author calls this a trick; the wedge business and alteration of BASIC's internal variable storage in the next section must be a veritable magic show! (The portability of such a scheme must be practically nil.)

■ The section beginning "A wedge is a method . . . " rapidly descends into an area which involves having access to (and being more than a little familiar with) the source code to one's BASIC interpreter, something that even *COMPUTER LANGUAGE*'s elite readership might find it hard to come upon.

■ Along those lines: If you are going to delve so deeply into the black box of the language's system software as to name specific interpreter source code labels (CHARGOT) and describe the interpreter's internal memory management scheme, then we are definitely not talking about most BASICs but rather about one specific implementation. In this case you really must state which BASIC it is to be of more than minimal curiosity value to anyone. (I suspect the author refers to one of the 8086/8088 BASICs due to the lack of a stack area in Figure 2. Those implementations probably use the separate stack segment for such purposes.)

■ Listing 3, line 2050, is a pretty obscure way of saying the old standard:

```
2050 IF INKEY$ = "" THEN 2050
```

Or, if you dislike *GOTO*s (as I suspect is the case here), then you could use:

```
2050 WHILE INKEY$ = "" : WEND
```

Not to mention the fact that some language implementations (especially compiled ones) don't recognize your alteration of a looping structure's control variable from within the loop.

■ Most Microsoft BASICs (and several others as well) will properly handle recursive *GOSUB*s due to their use of a stack internally to manage the depth of nesting and the RETURN addresses. The problem is the need for dynamic local variables in each reinvokation of the routine (rather than the static global variables BASIC provides). This may be simulated by means of arrays for each variable with a global "nesting-depth" counter to be the subscript for them all, thus providing a sort of variable stack somewhat similar to the internal handling of such things by ALGOL-like languages.

An example of this in Microsoft 8-bit CP/M BASIC is available on the *COMPUTER LANGUAGE* Bulletin Board Service or CompuServe account. It runs without change on TRS-80 Models I, III, and IV, as well as PC-DOS and MS-DOS versions of BASIC/BASICA/GWBASIC.

Also, you covered SNOBOL a few issues back (Premier issue, pp. 65-68). I saw in *PC TECH Journal*, and later purchased, a really nice implementation by Viktors Berstis for the IBM PC for only $39.95, plus postage and handling. (Minnesota SNOBOL4, Berstis International, P.O. Box 441, Millwood, New York, N.Y. 10546)

I really enjoy your efforts and appreciate a magazine which makes me think instead of feeding me product reviews of dozens of things I'll never need and beginners' guides to choosing a modem, and the like. Keep up the good work.

Chuck Somerville
Dayton, Ohio

Author Hugh Aguilar responds: I would like to thank Mr. Somerville for reading my article with such interest. He has pointed out some valid typographical errors for which I take full responsibility. The English algorithm is submitted as follows:

```
A. If DISKS is greater than zero, then
      solve by:
   A.1 Moving DISKS-1 disks from
          FROM to OTHER
   A.2 Moving the DISKS numbered
          disk from FROM to
          DESTINATION
   A.3 Moving DISKS-1 disks from
          OTHER to DESTINATION
B. End.
```

Also see Listing 1 (to replace Listing 2 in the article), complete with plus and equal signs, and Figure 1 (to replace Figure 3).

If it makes you feel any better, Mr. Somerville, I used the article money for a word-processor to replace my typewriter.

It is important to differentiate between tricks and techniques. Techniques are portable between languages. Structured writing and the evolving design of compilers is an effort to close the gap between the technique in the mind and the source on the screen. Because perfection in this effort hasn't been reached, application-specific tricks are necessary.

In BASIC it is necessary to simulate such things as WHILE . . . REPEAT, DO . . . UNTIL, *and recursive procedures—which are an integral part of real languages. I picked up a good trick for simulating the* DO . . . UNTIL *loop in* BYTE *a few years ago. It works something like this:*

```
for 1 = 0 to -1 step -1
        (block)
1 = (conditional expression): next 1
```

This works when Booleans are represented as a negative one or a zero. I discourage the use of the INKEY$. *Functions shouldn't be affecting globals or I/O as it*

```
let DISKS= DISKS-1: let A$= OTHER$: let OTHER$=
DESTINATION$: let DESTINATION$= A$: gosub HANOI:
let DISKS= DISKS+1: let A$= DESTINATION$:
let DESTINATION$= OTHER$: let OTHER$= A$
```

Listing 1.

```
                                      ,VS          ,AS,AE       ,SS,EM
BASIC    , variables, arrays, old     , input      , garbage    , output    , strings
program                       VS, AS    values                    names
                              AE, SS    without                   without
                              & EM      names                     values
```

Figure 1.

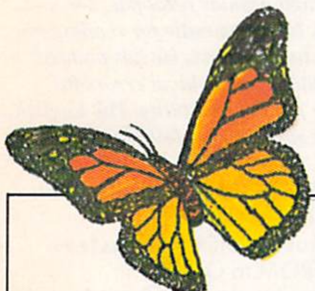
does. This can cause bugs when expressions are processed in parallel.

Procedures, complete with locals, are harder to simulate. Mr. Somerville's efforts can be seen in his HANOI program. My efforts are described in the article. An example of using a BASIC interpreter tricked in this way can be seen in my HANOI program. I feel that my method has several advantages.

■ *The chore of stack housekeeping is hidden from the BASIC programmer. This makes his or her job easier and simplifies the porting of the application to another language.*

■ *The stack housekeeping is done in assembly and hence is much faster.*

■ *Only locals are played around with in the block. The* GOSUB *sets up the reference points to the globals, and the* RETURN *puts the computed values into the actual variables. This has an advantage over the Pascal or ALGOL method of using the reference points (passed with* VAR*) like variables. In a multitasking environment the* RETURN *can wait for a signal from the supervisor before changing the actual global variables and relinquishing control.*



The Language of Artificial Intelligence

PROLOG V

Includes: *interpreter, sample Prolog programs, 122-page manual and primer, and custom-designed binder and slipcase.*

ONLY $69.95

Not Copy Protected

Selected by Japan's Fifth Generation ICOT Project to create machine-based intelligence, Prolog is the most widely used artificial intelligence language worldwide.

POWERFUL AND ELEGANT

With over 70 predefined predicates, PROLOG V complies with the spirit and syntax of Edinburgh Prolog as described in Clocksin and Mellish's classic *Programming in Prolog*, the book that set the de facto standard for Prolog.

IDEAL FOR CREATING:

- ☐ relational data bases
- ☐ expert systems
- ☐ natural language systems
- ☐ biochemical analysis
- ☐ automated design
- ☐ abstract problem solving
- ☐ mathematical logic

THE CHOICE OF UNIVERSITIES

Generous University site licenses have made PROLOG V the choice of Universities from New England to Southern California. Send your inquiry on University stationery, or call for details.

NO RISK OFFER

Examine the package and documentation at our risk for 30 days. Then if you aren't fully satisfied with the quality of the product, return PROLOG V with the distribution diskette still sealed, and we'll refund your money in full.

HARDWARE REQUIREMENTS
- ☐ PC-DOS/MS-DOS 1.4 or later
- ☐ 128 K RAM
- ☐ One disk drive

☐ PAYMENT ENCLOSED $ ____________
CA residents add 6% sales tax

☐ CHARGE MY: ☐ MasterCard ☐ Visa

Card No. ____________ Exp. Date ______

Signature ____________

Mr./Mrs./Ms. ____________
(please print full name)

Address ____________

City/State/Zip ____________

PHONE ORDERS:
(619) 483-8513

SHIPPING:
$ 5.00 U.S.
7.50 Canada
10.00 Carribean, Hawaii Air
20.00 Overseas Air

Allow 15 business days for personal and company checks.

COD orders not accepted.

CHALCEDONY SOFTWARE

5580 LA JOLLA BLVD.
SUITE 126 C
LA JOLLA, CA
92037

CIRCLE 49 ON READER SERVICE CARD

Cultural influence

Dear Editor:

I very much enjoyed your interviews with Donald Knuth and Niklaus Wirth. It is interesting that the differences between them are not very great. I find myself more in sympathy with Knuth, probably because I learned programming on the job rather than in school. Very early on, however, I learned the theory of structured programming and it has served me in good stead.

Now that I have learned Pascal I am far more fond of it than my earlier languages, mainly because it makes possible very clear expression of structured algorithms, as Wirth intended. One thing aggravates me about Pascal, however—and this is exactly to the point that Knuth raises, the need to write for other people to read, not just for the computer to execute—namely, the need to look to the very end of the written program to find the topmost level of control structure. This may be due to Wirth's design of a single-pass compiler, or perhaps it is a cultural reflection of the Germanic tendency at the very end of the sentence the verb to put, but it surely does not make for readable code!

I much prefer to put the main level of control first in the text, then the major subroutines, then the more detailed levels, etc., followed by utility routines that are called from lots of other places. I sure wish Pascal would let me do this.

Thanks for a stimulating and interesting pair of articles.

Bill Meacham

Parlez-vous Forth?

Dear Editor:

As a newcomer to both the Macintosh and MacFORTH, I read with interest "Hashing Out Forth with Charles Moore" (Mar. 1984, pp. 19-24). Ken Takara's question—"Do you have a language that you particularly like for its expressive qualities, something beyond the usual claims of efficiency or mere maintainability?"— brought to mind the following possibilities for the Forth language that I think few other languages could accommodate so easily.

I live in a part of Canada where the principal language is French. This means that when I am scouring the local libraries for books to help learn programming, I often end up reading books published in French. It has occurred to me that it must be difficult for non-English speaking people to learn computer programming when the key words of the major languages are in English.

Given the compactness of Forth and the fact that many of the words are just character symbols (@, !, :, etc.), it would be a relatively easy task to produce a Forth equivalent in almost any language.

Synonyms can easily be created for most words just by a simple redefinition (Echange swap or maybe Sortie exit). This makes the word available in users' native tongue but leaves the original English word available if they should care to use it as they grow more familiar with Forth (or for other users in a multiuser environment).

The character words could just be assigned a new pronunciation (for example, @ might become Chercher in French). The more complex words would require some redefinitions within the Forth kernel, but I don't think this poses any problem.

Of course, in any application the higher level words would be given descriptive names in the user's native language, as happens now.

Apple Computer has recognized the need to make its products more adaptable to people all over the world. The ability of the Macintosh to be reconfigured to foreign character sets and the availability of the keyboard in many international versions makes the personal computer a lot more personal to a lot more people.

MacForth is a very creative environment for programming the Mac. With Forth's adaptability and Apple's continued interest in the educational use of computers, perhaps for the first time students everywhere will learn to program in their native tongue.

This language adaptability also has some interesting prospects given the development of voice-activated computers. One problem with this type of communication is that each of us speaks with a slightly different accent or pronunciation. A set of Forth synonyms might be created by each individual as he or she speaks the commands into a microphone. These synonyms would be phonetic profiles of an individual's pronunciation of a word and would be handled internally by the Forth architecture in much the same way as ordinary word definitions. Thus a user could "teach" the computer to understand an accent just as we humans are able to adapt our hearing to comprehend a wide range of accents.

I believe that this ability of Forth to adapt to individual requirements, even to evolve into different native languages, gives it expressive qualities that are unique.

Grant Corriveau
Pierrefonds, Que.

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.

RDS™

RELATIONAL DATABASE SYSTEMS, INC.

CIRCLE 77 ON READER SERVICE CARD

WALTZ LISP

The universal, super-efficient Lisp
for PC-DOS, MS-DOS, CP/M-86 and CP/M-80 systems.

Waltz Lisp is a very powerful and complete implementation of Lisp. It is similar to *Franz* (the Lisp running under *Unix*), and is substantially compatible with *MacLisp* and other mainframe Lisps.

Ultra fast. In independent tests, Waltz Lisp was up to twenty(!) times faster than competing microcomputer Lisps.

Easy to use. Built-in WS-compatible full screen program editor. Full debugging and error handling facilities are available at all times. No debuggers to link or load.

Practical. Random file access, binary file support, and extensive string operations make Waltz Lisp suitable for general programming. A text-file difference program and other utilities are included in the package.

Full Lisp. Functions of type lambda (expr), nlambda (fexpr), lexpr, macro. Splicing and non-splicing character macros. Full suite of mappers, iterators, etc. Long integers (up to 611 digits). Fast list sorting using user defined comparison predicates. Built-in prettyprinting and formatting facilities. Over 250 functions in all.

Flexible. Transparent (yet programmable) handling of undefined function references allows large programs to reside partially on disk at run time. Optional automatic loading of initialization file. User control over all aspects of the system. Assembly language interface.

Superbly documented. Each function is described in detail. The 300+ page manual includes an exhaustive index and hundreds of illustrative examples.

Order Waltz Lisp now-and receive *free* our
PROLOG Interpreter
Clog Prolog is a tiny (but very complete) Prolog implementation written entirely in Waltz Lisp. In addition to the full source code, the package includes a 50 page Clog Manual.

16-bit versions require DOS 2.x or CP/M-86 and 90K RAM (more recommended). Z-80 version requires CP/M 2.x or 3.x and 48K RAM minimum. Waltz Lisp runs on hundreds of different computer models and is available in all disk formats.

$169*

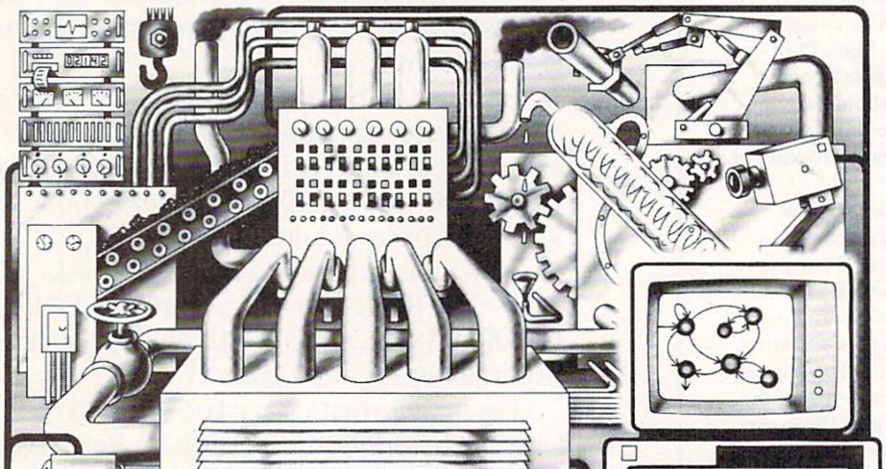
*Manual only: $30 (refundable with order). Foreign orders: add $5 for surface mail, $20 for airmail. COD add $3. Apple CP/M, hard sector, and 3" formats add $15. MC/Visa accepted.

For further information or to order call
1-800-LIP-4000 DEPT. 30
In Oregon and outside USA call 1-503-684-3000

PRO CODE™ INTERNATIONAL
15930 SW Colony Pl.,
Portland, OR 97224

CIRCLE 73 ON READER SERVICE CARD

Csharp Realtime Toolkit



Realtime on MSDOS? Csharp can do it! Get the tools without operating system overhead. Cut development time with C source code for realtime data acquisition and control. **Csharp** includes: graphics, event handling, procedure scheduling, state system control, and interrupt handling. Processor, device, and operating system independent. **Csharp** runs standalone or with: MSDOS, PCDOS, or RT11. **Csharp** runs on: PDP-11 and IBM PC. **Csharp** includes drivers for Hercules and IBM graphics boards, Data Translation and Metrabyte IO boards, real time clock, and more. Inquire for Victor 9000, Unix, and other systems. Price: $600

SYSTEMS GUILD

Systems Guild, Inc., P.O. Box 1085, Cambridge, MA 02142
(617) 451-8479

CIRCLE 27 ON READER SERVICE CARD

B+ trees, B++ trees, and statistics in AI

By Namir Clement Shammas

In last month's column, we began a discussion on searching techniques that employ tree structures. We indicated that binary trees were suitable for memory-based searches. We also introduced the B-tree structure, which is used to perform a disk-based search. We ended our discussion by pointing out that B-tree searching has its shortcomings. While searching for single keys is still efficient, it is somewhat clumsy to read a number of keys in sequence.

This month I want to discuss the B+ tree and its improved search strategy. B+ tree structures are used by a number of popular software products, such as dBase III, Borland's Turbo Toolbox, and many data management utility software libraries. In addition, I will present an enhancement of the B+ tree itself, which we will name the B++ tree.

Modification of the B-tree was the result of a need to reinforce the ability to more easily obtain a complete or partially sorted list, including locating a certain sought key and then asking for the next or previous keys. This procedure dictates that the leaf pages become doubly linked lists, allowing their traversal to either the left or right. Simultaneously, we need to have copies of all the keys inserted in the tree's leaves. Keys that are located in node pages will be duplicated in the leaves. This is the first major distinction of the B+ tree: doubly linked leaves containing all entered keys.

The second distinction comes from realizing that since node keys are duplicated in the leaves, we can strip the record data pointers from the node pages. This makes the data structure of node pages different from that of the leaves. By contrast, the B-tree has the same structure for both page types.

Figure 1 shows the B+ tree. Notice that keys in higher nodes are duplicated in each lower node. This duplication is necessary to locate the proper leaf containing a sought key.

Searching through the B+ tree begins at the root page. The keys are read and compared with the search key. The outcome of the comparison normally leads to the next page node. There the same key comparison is carried out to select yet another page node. Eventually the obtained pointer will lead to a leaf page. A final search in the leaf will determine whether or not the sought key exists. Thus, every search in a B+ tree is equivalent to the worst case search in a B-tree. However, this fact is not regarded as a serious drawback. Some people even praise the consistency of the real time involved in searching!

Listing 1 shows the PPL code for searching in a B+ tree. The main *Search* procedure calls upon *SearchNode* and *SearchLeaf* to scan the two different types of key pages.

Growth of a B+ tree and a B-tree are similar. Initially, there is one empty page. Keys are added and fill out the page. When an attempt is made to add a key to a full page, the page keys read in memory and the new key is inserted in the key list so that a perfect sorted order is maintained.

Next, the median key is selected, dividing the list into two halves. The lower half is written back to the old page, while the upper half is written to a newly created page. Pointers are used to establish the double link between the leaves. The median itself is stored as the last item of the original page. A copy of the median key (without the data record pointer) is stored in a new, higher-level node page. This makes the B+ tree grow by one level. The page containing the median key becomes the new root page. A comparison of the median keys will guide the search toward either leaf page.

This scenario takes place during the early growth stages of the B-tree. When more keys are added, they are inserted in the leaf pages. As each leaf page becomes full, it is split into halves and the median key inserted into the last location in the leaf pointing to the parent node page. If the parent node page becomes full, the same operation is carried out, resulting in two new half-full nodes. Their median is inserted or used to create a new parent node page.

Deleting from B+ trees is more complex than deleting from a B-tree. If the key deleted occurs in a leaf page only, then the operation is straightforward. Otherwise, we have two choices to make regarding the deletion of keys from node pages:

■ Delete the key from the leaf page only. Keep the keys in the node pages, but mark their status. Marking can be done by altering a status flag during the search to locate the key to be deleted. Decide on a criteria for packing the B+ tree.
■ Delete all occurrences of the key. This will involve some major rearranging in the node pages.

Now let's turn our attention to the B++ tree. Its structure serves to limit the duplication of

B+ tree: A partial view

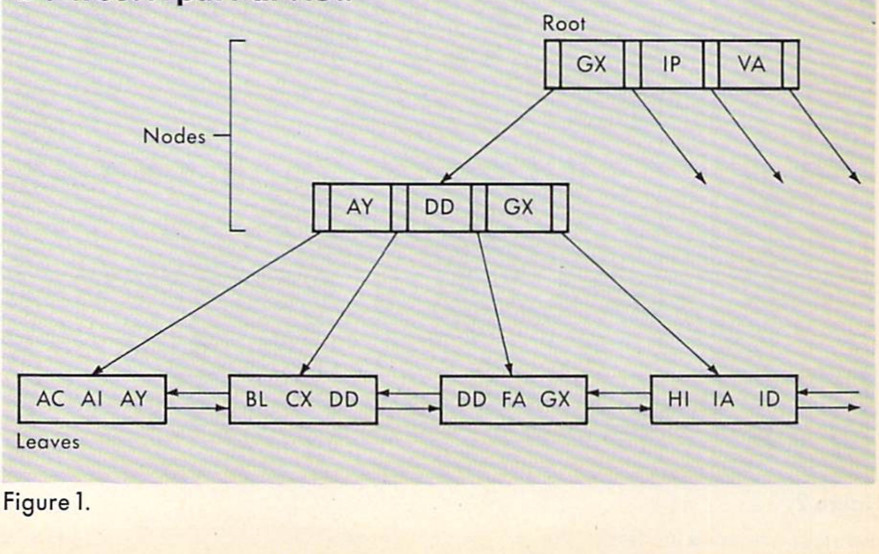


Figure 1.

keys and make deleting keys more simple. The B+ + tree is different from the B+ tree in the following respects:
■ Each key in the node pages has only one duplicate, located in the leaf pages.
■ The last stored key in each leaf is dedicated to duplicating node keys.
■ Zoom pointers are attached to each node key. They point directly to the leaf page containing the key with its record data pointer.
■ Nodes are doubly linked lists.
■ Searching in a B+ + tree is not necessarily equivalent to the worst B-tree search. With a B+ + tree you need one more disk access than with a comparable B-tree search.

When dealing with leaf pages, we declare their capacity, MAX_KEY, in terms of keys that are not duplicated in nodes. We deliberately save the (MAX_KEY + 1) position to store the duplicate of a node key.

Each node page is supplied with zoom pointers. If a search key equals a node key, then it is pointless to continue traversing node pages at lower levels. The zoom pointer has the address of the leaf page containing full information about the searched key. The latter is systematically located at the last "hidden" position in the leaf page. Figure 2 shows the B+ + tree.

B+ + tree and B+ tree searching are similar, with one exception: once the search key matches a node key, we use the zoom pointer to locate the sought leaf page and recall the last element in the leaf. Listing 2 shows the PPL code for searching in a B+ + tree.

Growth of the B+ + tree is very similar to that of the B+ tree, with some differences, such as additional zoom pointers in node pages. Allowing duplicate keys requires establishing double links between all pages. This is the price to pay for incorporating zoom pointers.

When a leaf page is reached via a zoom pointer, its parent is unknown. Knowing the parent is not required until the leaf is full and we need to split the page and insert a copy of the median key into the parent node. Thus each leaf page must, in turn, know its parent. A similar argument is used for node pages also. In case of overflowing keys, each node page must be able to point to its parent.

Deleting a B+ + tree key involves removing it from the leaf pages and any node page that duplicates it. This requires less node rearrangement than a B+ tree.

Since this issue is dedicated to artificial intelligence, a subject of fascination to many of us, I want to discuss some simple aspects of using statistics in heuristic systems.

During the last decade, much of the programming I've done has revolved around the use of regression analysis in R&D. The major purpose of this work was to study the correlations between observed variables in an attempt to obtain mathematical models that allow for future predictions.

Attempting to determine the best model is much like detective work. Statisticians have devised simple and limited methods to select the best models. Complications arise due to two factors: applying mathematical transformations to observed data and the varying number of terms in a mathematical regression model. There is a vast number of combinations for the candidate models.

I will discuss the simplest case for a regression system that learns from experience. This type of system stores the results of regression calculations and builds a performance history for competing regression models.

Listing 3 shows the PPL code for the system used. The program is designed to consider a simple linear regression between two variables. For simplification, only linearized models are considered, each having a slope and intercept, but applying different mathematical transformations.

The system is designed to contain a fixed number of competing models. This is another simplification. The coefficient of determination is used to indicate the goodness of fit. Its values range from zero (meaning no correlation exists) to one (meaning a perfect correlation is obtained).

Every time the program runs, it reads two data files. The first contains the performance history of the models. The second contains the observations to be processed. The program will process the data for each regression model.

All models are used for the first few sets of data, because it is too early to disqualify any model. The coefficient of correlation is obtained for each model and is used to update the performance history. A value close to unity signals good performance. For the first few sets of data (three in our case) the program stops after all the models are processed. It is too early to start screening the models.

On the other hand, when enough data sets have been processed, we begin model screening. We pick the best model, one with the highest average coefficient of determination. We compare all other models to the selected one. Using a simple statistical test, we determine, at a certain probability level, whether any other model's coefficient of determination has a statistically different value. If it does, that model is disqualified and barred from further consideration. This is done by altering the *MAP()* identifier. After this is done, the loop that processes the data will bypass all disqualified models.

This process will continue until a single model is found to fit. How soon we obtain the model that fits depends on many factors:
■ The nature of the system studied. If simple linearized models are not really suitable, then the screening process may take much longer to produce a single, reliable model. Failing to do so is also meaningful. It tells us that there is a more complex correlation between the observed variables.
■ The amount of data in each set processed.
■ The accuracy of the processed data. Poor data will throw the system (and any human) off.

These factors reflect a simple heuristic system that learns by storing intermediate results and uses statistical methods to draw conclusions. Its limitations include the inability to vary the number of terms in the mathematical regression model and to deduce what mathematical transformations should be used. This area will be discussed in a future column. ■

B + + tree: A partial view

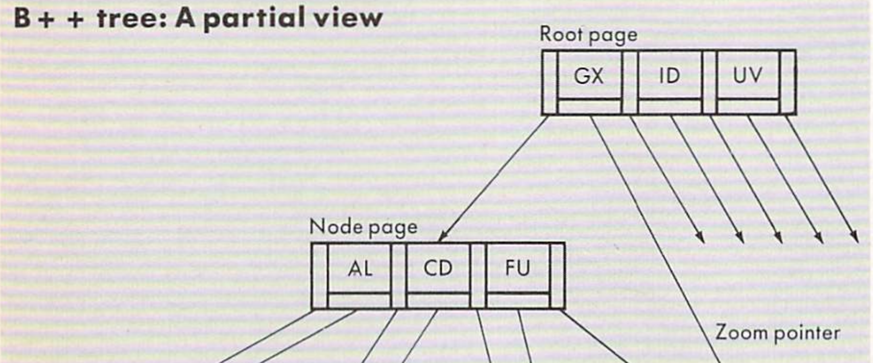


Figure 2.

```
PPL code for searching in a B+ tree

-------------------------------------------------------B+ tree

-- DATA TYPE DECLARATION, Pascal style

Leaf_Rec = RECORD
        Leaf_Left, Leaf_Right, Count_Leaf_Key: INTEGER
        Leaf_Key : ARRAY[1..MAX_KEY+1] OF Key_Data
        Leaf_Ptr : ARRAY[1..MAX_KEY+1] OF INTEGER
END RECORD

Node_Rec = RECORD
        Count_Node_Key : INTEGER
        Node_Key : ARRAY[1..MAX_KEY+1] OF Key_Data
        Node_Ptr : ARRAY[1..MAX_KEY+1] OF INTEGER
END RECORD

-- VARIABLE DECLARATION

Leaf : Leaf_Rec
Node : Node_Rec
ROOT, HEIGHT, MAX_KEY, NUM_PAGE : INTEGER


PROCEDURE Search(Soughtkey : Key_Data
                 ROOT, HEIGHT : INTEGER
                 VAR Found : BOOLEAN
                 VAR SoughtLeaf, Sought_Loc : INTEGER
                 VAR Leaf : Leaf_Rec)

-- Search procedure for B+ tree


Found = FALSE
IF HEIGHT > 0 THEN
    INITIALIZE: None
    LOOP
    -- loop will conduct gradual descent in B+ tree
    BEGIN IF HEIGHT <= 1 THEN EXIT END IF
        READ "B+TREE_Key_File",ROOT,Node
        SearchNode(SoughtKey,Node,Found,Sought_Loc)
        ROOT = Node.Node_Ptr[Sought_Loc]
        HEIGHT -= 1
    END LOOP
    TERMINATE: SoughtLeaf = ROOT
               READ "B+TREE_Key_File",SoughtLeaf,Leaf
               SearchLeaf(SoughtKey,Leaf,Found,Sought_Loc)
END IF
END Search

-----------------------------------------------------------
```

Listing 1. *(Continued on following page)*

```
PROCEDURE SearchNode(SoughtKey : Key_Data;
                     Node : Node_Rec;
                     VAR Found : BOOLEAN;
                     VAR Sought_Loc : INTEGER)

BEGIN

    IF SoughtKey < Node.Node_Key[1]
    THEN
      Found = FALSE
      Sought_Loc = 1

    ELSE
     INTIALIZE: Sought_Loc = Node.Count_Node_Key
     LOOP
     BEGIN IF (SoughtKey >= Node.Node_Key[Sought_Loc]) OR
              (Sought_Loc <= 1) THEN EXIT END IF
           Sought_Loc -= 1
     END LOOP
     TERMINATE: Found = (SoughtKey = Node.Node_Key[Sought_Loc])
    END IF
END SearchNode
```

Listing 1. *(Continued from preceding page)*

```
PPL code for searching in a B++ tree

-----------------------------------------------------------B++ tree

-- DATA TYPE DECLARATION, Pascal style

Leaf_Rec = RECORD
        Leaf_Left, Leaf_Right, Count_Leaf_Key, Node_Above : INTEGER
        Leaf_Key : ARRAY[1..MAX_KEY+1] OF Key_Data
        Leaf_Ptr : ARRAY[1..MAX_KEY+1] OF INTEGER
END RECORD

Node_Rec = RECORD
        Count_Node_Key, Parent_Node : INTEGER
        Node_Key : ARRAY[1..MAX_KEY+1] OF Key_Data
        Node_Ptr, ZoomPtr : ARRAY[1..MAX_KEY+1] OF INTEGER
END RECORD

-- VARIABLE DECLARATION

Leaf : Leaf_Rec
Node : Node_Rec
ROOT, HEIGHT, MAX_KEY, NUM_PAGE : INTEGER

-------------------------------------------------------------------

PROCEDURE Search(Soughtkey : Key_Data
                 ROOT, HEIGHT : INTEGER
                 VAR Found : BOOLEAN
                 VAR SoughtLeaf, Sought_Loc : INTEGER
                 VAR Leaf : Leaf_Rec)
```

Listing 2. *(Continued on following page)*

```
Found = FALSE
IF HEIGHT > 0 THEN
    INITIALIZE: None
    LOOP
    BEGIN IF HEIGHT <= 1 THEN EXIT END IF
        READ #2,ROOT,Node
        SearchNode(SoughtKey,Node,Found,Sought_Loc)
        IF Found THEN HEIGHT = 1;  ROOT = Node.Zoom_Ptr[Sought_Loc]
                 ELSE HEIGHT -= 1; ROOT = Node.Node_Ptr[Sought_Loc]
    END LOOP
    TERMINATE:  SoughtLeaf = ROOT
                READ #2,SoughtLeaf,Leaf
                IF NOT Found THEN SearchLeaf(SoughtKey,Leaf,Found,Sought_Loc)
                             ELSE Sought_Loc = MAX_KEY + 1
END IF
END Search

-- Procedures SearchNode and SearchLeaf are identical to those used
-- with the B+ tree.
```

Listing 2. *(Continued from preceding page)*

```
PPL code for a heuristic statistical system.

PROGRAM SMART_SYSTEM

-- Program to study the best model to correlate observed
-- variables X and Y.  The models used are of the following
-- general type:

--     f(Y) = intercept + slope g(X)

-- where f(Y) is some function of variable Y.
-- where g(X) is some function of variable X.

-- Performance history file contains the following information
--     MAX_MODEL : the maximum number of models compared
--     Remaining_Models : The number of remaining models
--     MAP(): A string containing MAX_MODEL characters to map the status
--            of each model.  If position i has "Y" then the model is
--            still considered.  Otherwise it is disqualified.
--     Model records composed of the following:
--         Sum_R2 : Sum of the coefficient of determination values
--         Sum_Sqr_R2 : Sum of the squares of the coefficient of
--                      determination values
--         Model_Name : A string containing the model name.

-- N is the number of data sets processed
-- X() and Y() are the arrays for the observations.
```

Listing 3. *(Continued on following page)*

```
BEGIN
    Read performance history file
    IF Remaining_Models = 1 THEN DISPLAY "One model, program stopped"
                                 Halt program
    END IF
    Read observed data file
    N += 1
    INITIALIZE: None
    LOOP <Model>
    BEGIN  For Model = 1 TO MAX_MODEL
        INITIALIZE: Set statistical summations to zero
                    i = 1
        LOOP <Data>
        BEGIN IF (i > Num_Data) OR (MAP(Model) <> "Y")
              THEN EXIT <Data> END IF
          CASE i OF
             WHEN 1 => Xreg = X(i); Yreg = Y(i)      -- Linear model
             WHEN 2 => Xreg = Ln(X(i)); Yreg = Y(i) -- Exponential model
             WHEN 3 => Xreg = X(i); Yreg = Ln(Y(i)) -- Logarithmic model
             WHEN 4 => Xreg = Ln(X(i); Yreg = Ln(Y(i)) -- Power model
          END CASE
          Update summations with values of Xreg and Yreg
          i += 1
        END LOOP <Data>
        TERMINATE: None
        Calculate Slope, intercept and coefficient of determination, R2
        Sum_R2 += R2; Sum_Sqr_R2 += R2 * R2
    END LOOP <Model>

    IF N > 3 THEN
        Best = index of model with highest average R2 value.
        INITIALIZE: Display "Best model is";Model_Name(Best)
                    Table_T = "Tabulated" Student-t value calculated
                               at (N-2) degrees of freedom and
                               selected probability level.
        LOOP <Compare>
        BEGIN For  Model = 1 to MAX_MODEL
            IF (Model <> Best) AND (MAP(Model) = "Y") THEN
                -- Calculate Student-t statistics
                Term1 = Sqrt(2/N)
                Term2 = Sqrt((Sum_Sqr_R2(Best) + Sum_Sqr_R2(Model)
                         - Sqr(Sum_R2(Best))/N - Sqr(Sum_R2(Model))/N)
                         / (2 * N - 2))
                CalcT = (Sum_R2(Best) - Sum_R2(Model))/(N * Term1 * Term2)
                IF ABS(CalcT) > Table_T THEN MAP(Model) = "N"
                                             Remaining_Models -= 1
                END IF
            END IF
        END LOOP <Compare>
        TERMINATE: Display all models still in competition
    END IF
END SMART_SYSTEM
```

Listing 3. *(Continued from preceding page)*

Slash Programming Time in Half!

With **FirsTime**™

- Fast program entry through single keystroke statement generators.
- Fast editing through syntax oriented cursor movements.
- Dramatically reduced debugging time through immediate syntax checking.
- Fast development through unique programmer oriented features.
- Automatic program formatter.

FirsTime is a true syntax directed editor.

FirsTime ensures the integrity of your programs by performing all editing tasks like moves, inserts and deletes along the syntactic elements of a program. For example, when you move an IF statement, FirsTime will move the corresponding THEN and ELSE clauses with it.

Even FirsTime's cursor movements are by syntax elements instead of characters. The cursor automatically skips over blank spaces and required keywords and goes directly to the next editable position.

FirsTime is a Syntax Checker

FirsTime checks the syntax of your program statements, and also:

- Semantics like undefined variables and mismatched statement types.
- The contents of include files and macro expansions.
- Statements for errors as they are entered and warns you immediately.

FirsTime is a Program Formatter

FirsTime automatically indents statements as they are entered, saving you from having to track indentation levels and count spaces.

FirsTime has Unique Features

No other editor offer these features:

The *Zoom command* gives you a top down view of your program logic.

The *View command* displays the contents of include files and macro expansions. This is valuable to sophisticated programmers writing complex code or to those updating unfamiliar programs.

FirsTime's *Transform command* lets you change a statement to another similar one with just two keystrokes. For example, you can instantly transform a FOR statement into a WHILE statement.

The *Move at Same Level command* moves the cursor up or down to the next statement at the same indentation level. This is very useful. For example, you can use it to locate the ELSE clause that corresponds to a given THEN clause or to traverse a program one procedure at a time.

FirsTime is Unparalleled

FirsTime is the most advanced syntax directed editor available. It makes programming faster, easier and more fun.

TO ORDER CALL (201) 741-8188

or write:

Spruce Technology Corporation

189 E. Bergen Place
Red Bank, NJ 07701

In Germany, Austria and Switzerland contact:

Markt & Technik Software Verlag
Munchen, W. Germany
(089) 4613-0

FirsTime is a trademark of Spruce Technology Corporation • MS-DOS is a trademark of Microsoft Corporation
IBM is a trademark of International Business Machines, Inc. • Turbo Pascal is a trademark of Borland International

CIRCLE 26 ON READER SERVICE CARD

FirsTime for **Turbo Pascal $\$74^{95}$**

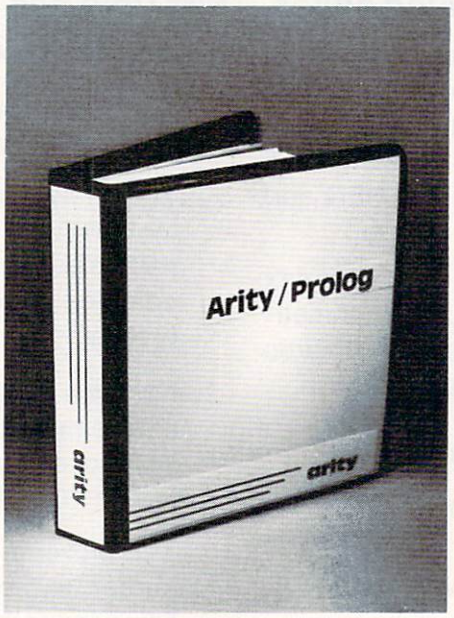
FirsTime for **MicroSoft & IBM Pascal $245**

FirsTime for **C $295**

More Power Than You Thought Possible

Arity offers the first serious implementation of Prolog for IBM personal computers. Arity/Prolog is a powerful, highly optimized, and extended version of the logic programming language Prolog. Imagine building software applications with a language that solves problems through deduction and logical inference. The task of creating complex programs is much faster and easier, resulting in lower development costs. Arity/Prolog is now in use in a wide range of applications in industry, business, research, and education. The solution—the *Arity/Prolog Interpreter:*

- Source level debugger
- Virtual databases, each with a workspace of 16 megabytes
- Floating-point arithmetic
- String support for efficient text handling



- Interface to assembly language and 'C'
- Text screen manipulation
- Integrated programming shell to MSDOS
- Comprehensive set of evaluable predicates
- Definite clause grammar support

Arity/Prolog Interpreter $495.00

Arity also offers the *Arity/Prolog Compiler and Interpreter,* a sophisticated development environment for building AI applications. Essential for producing fast, serious production code.

Arity/Prolog Compiler and Interpreter $1950.00

The *Arity/Prolog Demo Disk* is available for $19.95. ■ Arity/Prolog products run on the IBM PC, XT, AT, and all IBM compatibles. ■ To order, call (617) 371-2422 or use the order form below.

arity corporation 358 Baker Avenue, Concord, MA 01742

Name

Organization

Address

Quantity	Product	Unit Price	Total Price
	Arity/Prolog Compiler & Interpreter	$1950.00	
	Arity/Prolog Interpreter	$ 495.00	
	Arity/Prolog Demo Disk	$ 19.95	
		Subtotal	
		MA residents add 5% sales tax	
		Total Amount	

☐ Enclosed is a check or money order to Arity Corporation

☐ Please bill my ☐ Mastercard ☐ Visa ☐ American Express

Account #

Valid ____/____ to ____/____ signature

☐ Please send me more information about Arity and Arity/Prolog

arity 358 Baker Avenue, Concord, MA 01742

M-AD-C8

CIRCLE 48 ON READER SERVICE CARD

AI innovator Terry Winograd

By Doug Millison

Artificial intelligence pioneer Terry Winograd doesn't intend to put programmers out of work by designing the next generation of computer languages. "They'd just laugh," he says. "Programmers know you can't automate them out of existence." But while they may be spared the irony of being replaced by computers, programmers can expect a scramble to keep up as Winograd's research revolutionizes the state of the art of software.

Sitting in his computer science department office at Stanford University in Palo Alto, Calif., his desk overflowing with galleys of his most recent book, piles of newspaper clippings, and literature related to his work against military funding of university computer science research, Terry Winograd speaks quickly, hurrying to make his words keep up with the flow of his ideas. It's a level of excitement and enthusiasm he's maintained since he laid eyes on his first computer back in 1965.

The whiz kid from Colorado has been fascinated by numbers as long as he can remember. His first encounter with a computer came in college while he was working on his B.A. in mathematics. "There was a doctor at the local hospital who moved to our town from a larger university and brought with him a Control Data 160 with 4K of 12-bit memory, a paper-tape reader, and a paper-tape punch," he recalls. The doctor called Colorado College to see if they had any bright students who might be interested in working with the computer to help with his research project on cancer radiation therapy.

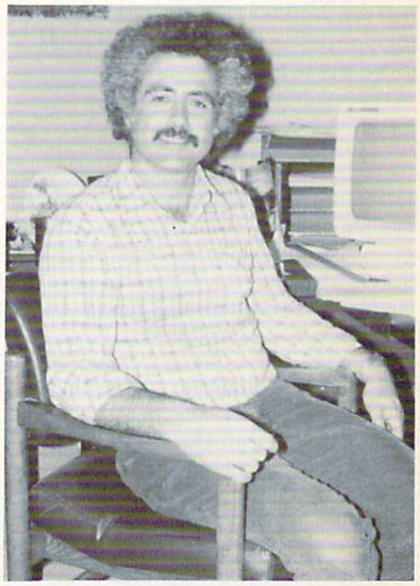
"I knew nothing about computers or programming," he says. "What I got was the machine and the instruction manual for the machine. I learned to program the thing by punching the console buttons. I used to have the repairman in every couple of weeks to repair the buttons, and I don't think he ever figured out what was going on."

His Ph.D. dissertation, published in 1970 out of the Massachusetts Institute of Technology, Cambridge, Mass., was a pioneering effort in the field, demonstrating how successful AI programs could be within a restricted domain. The dissertation centered on SHRDLU, a LISP program to take English commands and questions and give answers. Winograd wanted to design a program that would enable an imaginary robot hand and eye system to recognize, locate, and manipulate a universe of colored cubes and pyramids existing as a model within the computer itself. In SHRDLU, Winograd addressed the task of designing a system with which humans could communicate in natural language.

Responding to a request to place one block on top of another, SHRDLU would assess the universe of blocks on the table, develop a strategy, then execute it. The program could answer questions about its moves, carrying out the conversation in simple sentences and appearing to understand questions put to it, then generating appropriate responses. "SHRDLU was a milestone for me," Winograd says, "and I think it was for the lab and AI in general because it pulled together a lot of the ideas that had been floating around in AI into a very visible, concrete program."

His choice of SHRDLU as a name for the program reveals a subtle sense of humor and a streak of irreverence. SHRDLU happens to be the seventh through 12th most frequently used letters in the English alphabet, after E T A I O N. In earlier days of Linotype machines and hand-set type, keyboards were arranged in that manner instead of the QWERTY arrangement of today's typewriters and terminal keyboards. Instead of tediously going back to correct mistakes, typesetters would insert SHRDLU several times to jumble the line of text, counting on the proofreaders to throw out the line. Occasionally the proofreader would miss it, resulting in the absurd repetition of SHRDLU SHRDLU SHRDLU.

"I first saw it in *Mad* magazine. They started using SHRDLU as a nonsense name for some stupid character or something. I wanted a name for my program that wasn't an acronym, just a sequence of letters with no intrinsic meaning," he says.



At MIT, Winograd had found himself working in the most exciting arena in computer research. Notorious in the annals of the history of computing, the AI Lab at MIT harbored a generation of researchers whose work shaped the AI field we know today.

The PDP-6 in the lab had become the center of cult like devotion for those bitten by the bug. Because there was only one computer and a small number of terminals, competition became intense among those who wanted access. In his book *Hackers, Heroes of the Computer Revolution*, Steven Levy describes two opposing groups at the lab: the graduate students—disciples of the IBM batch-processing military research mentality, interested only in their applications programs and academics—and the hackers—renegades dedicated to exploring the limits of the machine in an unrestricted atmosphere of anarchistic freedom.

Winograd remembers straddling the fence. He was a graduate student working hard on his applications-oriented AI research for his Ph.D. But his ability to hack code gave him a grudging respect for the hardcore hackers, and they found

SHRDLU, his pioneering AI program, an "interesting hack."

"There was definitely the sense of 'Let's see what we can come up with' that led to everything from Space War to the artificial intelligence research I was involved in," he remembers.

Developments in the industry since those early days at MIT have rendered moot much of the hacker-vs.-computer-establishment debate. Although Winograd finds the hackers vs. IBM dichotomy far too simplistic to characterize what was happening at the MIT AI Lab, he says that real issues lurking behind that metaphor still haunt decision making in computer research policy today.

"The cost/benefit view says you take your resources and figure out what's going to be useful to somebody. On the other side is the idea that you find the people who are best qualified to play with the computers and tell them, 'Do whatever you can.' Nobody but the most extreme hackers would say the U.S. government should set up a billion dollar fund to let the hackers play," he observes.

Winograd agrees that hackers represented a useful reaction to the IBM mentality of regarding the machines with awe and deferring to their keepers as if to high priests of some arcane religious cult. But he sees a downside to hacker anarchism in its inability to meet society's needs for computer research.

"I think the negative side of the hacker thing is that it really doesn't connect with what it is that people can do and want to do with computers," says Winograd. "It's purely technology-driven: what can you make the machine do, whether or not it's relevant to anybody. We need to drive the technology from an understanding of the social situation, not just pursue technology for its own sake. Fortunately, computing now is cheap, you have anarchy by default. Anybody who can go buy a Macintosh or PC or whatever can start doing things, interesting things."

In those early days at the MIT AI Lab, the possibilities for truly intelligent machines seemed almost unlimited, if only enough resources were applied. The years since have seen changes in the field of AI itself and in Winograd's expectations for AI.

"Nobody can say, 'It's been 10 years and we now have this big wonderful AI program that does all these wonderfully intelligent things.' If you believe in AI, things look good, and if you don't believe in it, you see stagnation," he says.

Winograd sees two distinct and fundamental developments in AI. "First, there has been a recognition that there are a lot of salable things you can do that don't involve real intelligence," he observes. "There's been a mass diversion of effort away from the larger AI problem and into expert systems, front ends, things which are much more limited but which you can sell."

The second development concerns the atmosphere surrounding AI research. "I think among the main people in the field there is much more caution than 10 years ago. Now people like [Marvin] Minsky and [Roger] Schank say, 'This is very hard, it's going to take a long time, we really aren't there yet,' where 10 years ago they said 'We're going great, we're almost there.' There's been a definite shift of mood," Winograd says.

Winograd does not apply for or accept any Pentagon funding for his research. His view is that the military way of doing business and looking at the world has a negative impact on researchers and on society as a whole.

Many observers were deeply critical of the early claims of AI researchers. They challenged the belief that machines could think or be considered intelligent. They questioned the existence of any clear understanding of "intelligence" or "thinking" on which meaningful research could be based. In his book *What Computers Can't Do,* philosopher Hubert Dreyfus argued that the overoptimism about AI and expert systems is based on a misunderstanding of what it is that people do when they are "intelligent" or "expert." Winograd's colleague Joseph Weizenbaum elaborated a moral argument in *Computer Power and Human Reason*, contending that some tasks ought to be attempted only by humans.

Winograd began exploring the potential and limits of computers and how to distinguish what computers do from human thought. "I have really come much closer to a way of looking at AI like Weizenbaum or Dreyfus, as opposed to the gung-ho MIT AI Lab spirit," Winograd says.

With a background in linguistics as well as math and computers, Winograd was acutely aware of how the words and concepts we use shape the way we think about computers, how the metaphors we employ tend to channel and circumscribe the possibilities we envision for computers and for the people who are supposed to use and benefit from computers. He addresses these and other questions in a new book to be published this summer, *Understanding Computers and Cognition*. Co-author Fernando Flores, a former finance and economics minister in the Chilean government of Salvador Allende, had been instrumental in a large-scale project to apply cybernetic theory to practical management problems.

"To oversimplify, if I say, 'This machine is making judgments' and you want to know what the machine is really doing, I have to reply, 'It's doing these calculations.' Now, there's a pressure to say that these calculations constitute making judgments. This pushes out of the program and out of our notion of what it is the program should be doing, things which can't be formulated in calculations. We devalue the currency by saying 'This computer is teaching,' or 'This computer is making judgments.' This affects our expectations of what people should do when they do the same things," Winograd explains.

Underlying AI research is the redefinition of intelligence to include machines. Winograd warns this could change the way we deal with children and education. "If intelligence is defined as the ability to do problem solving, modeled as a heuristic search in a problem space, then education will focus on those things that increase rational problem-solving skills in areas where those skills are basic. Writing a computer program is a very good context for that," he explains.

The danger is that educators might shy away from contexts where rational problem solving does not work so well, which includes most interpersonal skills. "It turns out that rational problem solving is usable, but it doesn't tell you the whole story. It's like the old saying that if you have a hammer, then everything in the world looks like a nail," says Winograd.

Just as our understanding of computers changes our definition of intelligence, so our concept of the human mind itself has been reformulated to follow the computer model. Current brain research takes for granted a view of the brain as a complicated network of information-processing circuits.

Winograd has thought deeply about this "mind as machine" metaphor. In it he sees two different part-truths that have been combined. The first is the belief that the mind is mechanistic, with no soul or vital force. Given the state of religion and secular humanism in society today, this mechanistic view of the brain is more or less taken for granted. The second is that digital computers operate in a way that can be characterized elegantly as bytes, bits, transmission, logical choice, rational

Now Supports Microsoft 3.0

$149.95

Vitamin C

Windows that open, close, grow, shrink, move & scroll. Input validation,formatting, editing & processing. Help messages by field, by key word & from help file. Date & time math, attribute control, pull down menus, and more!

Now even your simplest applications can easily include features that you didn't used to have time or patience enough to tackle. Vitamin C is more than a library full of building blocks. It is a well planned, tightly woven set of high level functions for quick results PLUS low level routines for complete control. Complete source code, no royalties. Great manual with tutorial and reference. Sample programs too! For Microsoft 3.0, CI-C86, & Lattice C. Call for other versions, systems & products.

CALL TODAY ! Or send 149.95+ $3 shipping and handling. Texas add sales tax. MasterCard and Visa accepted.

Creative Programming
Box 112097
Carrollton, Tx 75011
(214)243-6197
Extended Technical support availab le.

CIRCLE 52 ON READER SERVICE CARD

INSIGHT™
EXPERT SYSTEMS

"INSIGHT is essentially the equivalent or better than any other tool available for the personal computer."

Paul Harmon, author of *Expert Systems, Artificial Intelligence in Business*

Turn your PC into an expert.

Give it Insight, or give it Insight 2. Both let you create knowledge base systems using any PC-compatible text editor.

Insight not only simplifies access to lots of information, it analyzes and offers solutions. For entry-level operators it's a perfect procedural training package to help build and implement knowledge base software.

Insight 2 is more than just an "expert." It's a knowledge base engineering tool with application capabilities. It can call up Pascal programs, read and write dBASE II® files, and its decision-making process can tie in directly to your existing databases. Run-only versions also can be developed and distributed.

Two unique packages from the same expert idea.

Level Five Research, Inc.

Insight™ ($95) and Insight 2™ ($485) run on the IBM® PC, DEC® Rainbow, and Victor® 9000.

4980 South A-1-A   Melbourne Beach, Florida 32951   (305) 729-9046

CIRCLE 64 ON READER SERVICE CARD

Macintosh Programmer's Library

Affordable, Reliable Tools for Programmers

From the author of Pascal/MT+ Tardis Software presents the Mac Programmer's Library

A collection of tools for the serious Mac programmer. Tools which save you hours of work written by an experienced Macintosh programmer. Tardis Software is committed to giving you reliable tools at a good price. Order 3 and save 10%, order all 5 and save 20%!.

The Macintosh Programmer's Library current contents:

- FastFinder - command-line oriented user-interface.
- ToolPak 1 - diff, search, dump ASM xref, librarian
- ToolPak 2 - c beautifier, tool subroutines (C source) and C xref
- C-leaner - expert system program, tells you where and how to optimize existing code
- MacMake - program maint. utility, simplifies re-builds

All are $49.95! and are AVAILABLE NOW

MC/Visa. COD's and POs add $5 for handling. Price incl. UPS ground, Blue Label add $5

Tardis Software
2817 Sloat Road
Pebble Beach, CA
(408) 372-1722

CIRCLE 82 ON READER SERVICE CARD

problem-solving, and information flow. Therefore, the equation goes, the mind must be that kind of machine.

Winograd finds it plausible that there might be a strictly mechanical explanation for what the brain does. But that doesn't automatically lead to the conclusion that brain function corresponds to the way a computer is structured into information-processing elements. "You can't say, 'This element sent that message of so many bits to that element, because the brain isn't constructed using those principles,'" he explains.

Winograd feels the current vogue for the left-brain/right-brain model is a partial retreat from the mind as machine metaphor. Those who believe that the mind is basically a computer now claim only the logical left half of the brain is, leaving the intuitive right brain for those fuzzy-minded individuals who can't swallow the whole machine.

Winograd sees a further danger accompanying the implementation of expert systems. Computers are increasingly seen to be infallible, and people, especially bureaucrats, find it easy to hide behind the computer. Responsibility is shifted away from the people who are responsible for policy and onto the computer in which decision-making power has been embodied. Problems arising from interactions with expert systems could lead to legal disputes, as in trying to ascertain who is responsible for the mistakes made by a medical AI program or the disasters resulting from a doctor's decision not to follow the expert system's advice.

The example of a medical expert system is a good one to illustrate Winograd's current thinking on AI. End-user perception of the system turns out to be just as important as what the system actually does.

The insights of many physicians could contribute to a system called a "physician expert system" or an "automated textbook." Winograd says people want to believe in the myth of computer infallibility, to believe computers are becoming more intelligent than their creators and that their speed and access to memory banks can replace human decision making in complex situations. Thus, people will have a tendency to look to the machine to replace the physician, investing the machine with the confidence and trust essential to the relationship we enjoy with human physicians.

"If you ask, 'Would you like this automated textbook, or would you rather have this doctor-expert machine?' the intelligent machine is so much more glorious sounding and has so much more sales appeal that it is the thing people want. But what you get can only be the opinions of various doctors, not the knowledge of medicine," Winograd says. "I think expectations will be dashed. When people say, 'OK, I want a doctor,' they will try the machine and find out that it isn't a doctor but that it does give them useful information. Then other companies will come in behind and say, 'Look, here's a machine which is organized much better to give you what you really need.' "

Winograd's current research builds on this more realistic appraisal of AI and on his interest in natural language syntax and semantics. He is acutely aware of the technical complexities of natural language processing and the subtle interpretations people make which must somehow be translated if we are to interact with computers using ordinary language. "The main problem I am interested in is how you can make a formal analysis of linguistic structure, both in terms of meaning and grammar, which will give you the necessary framework for computer languages," he says.

This approach led to work that continues the line of computer language evolution that Winograd traces from machine language to assembly language, to languages like FORTRAN, and on to fourth generation languages such as MAPPER. "What I'm doing I don't like to call programming languages," he says. "I haven't got a name I'm totally happy with, but at the moment I'm calling them system development languages."

He researches these new languages on a Xerox LISP machine. "LISP is obviously not a host for widespread applications on today's machines, but I start at the highest level of interactive facilities with it," he explains.

CP/M-80 C Programmers . . .

Save time

. . . with the BDS C Compiler. Compile, link and execute *faster* than you ever thought possible!

If you're a C language programmer whose patience is wearing thin, who wants to spend your valuable time *programming* instead of twiddling your thumbs waiting for slow compilers, who just wants to work *fast,* then it's time you programmed with the BDS C Compiler.

BDS C is designed for CP/M-80 and provides users with quick, clean software development with emphasis on systems programming.

BDS C features include:

- Ultra-fast compilation, linkage and execution that produce directly executable 8080/Z80 CP/M command files.
- A comprehensive debugger that traces program execution and interactively displays both local and external variables by name and proper type.
- Dynamic overlays that allow for run-time segmentation of programs too large to fit into memory.
- A 120-function library written in both C and assembly language with full source code.

Plus . . .
- A thorough, easy-to-read, 181-page user's manual complete with tutorials, hints, error messages and an easy-to-use index — it's the perfect manual for the beginner *and* the seasoned professional.
- An attractive selection of sample programs, including MODEM-compatible telecommunications, CP/M system utilities, games and more.
- A nationwide BDS C User's Group ($10 membership fee — application included with package) that offers a newsletter, BDS C updates and access to public domain C utilities.

Reviewers everywhere have praised BDS C for its elegant operation and optimal use of CP/M resources. Above all, BDS C has been hailed for it's remarkable *speed.*

BYTE Magazine placed BDS C ahead of all other 8080/Z80 C compilers tested for fastest object-code execution with all available speed-up options in use. In addition, BDS C's speed of compilation was almost *twice* as fast as its closet competitor (benchmark for this test was the Sieve of Eratosthenes).

"I recommend both the language and the implementation by BDS very highly."
Tim Pugh, Jr.
in *Infoworld*

"Performance: *Excellent.*
Documentation: *Excellent.*
East of Use: *Excellent.*"
InfoWorld
Software Report Card

". . . a superior buy . . ."
Van Court Hare
in *Lifelines/The Software Magazine*

Don't waste another minute on a slow language processor. Order your BDS C Compiler today!

Complete Package (two 8" SSSD disks, 181-page manual): **$150**
Free shipping on prepaid orders inside USA.
VISA/MC, COD's, rush orders accepted.
Call for information on other disk formats.

BDS C is designed for use with CP/M-80 operating systems, version 2.2. or higher. It is not currently available for CP/M-86 or MS-DOS.

BDSoftware

BD Software, Inc.
P.O. Box 2368
Cambridge, MA 02238
(617) 576-3828

CIRCLE 4 ON READER SERVICE CARD

He begins by looking at the "layers of thinking" in the process of going from a task he wants a computer to handle to a piece of code running on the computer system. "Starting from the outside, you first decide which aspects of the situation are going to be formalized and how. You need an abstraction to be the basis of the program before any details of the code or data structures, or anything else. You map the fuzzy, open-ended view of the world onto a set of terms, measurements, and values which then can be represented on a computer," he says.

He wants to bridge the gap between the person who specifies which functions and tasks the computer needs to perform and the translation of those requirements into code.

"I'm interested in what a language or set of languages would look like aimed toward that outside end instead of in toward machine execution," he says. "They would allow you to state things at a level of abstraction one step further from machine execution into the conceptual domain of what you're trying to do."

If the languages are successful, Winograd says one result might be the development of automated programming. Once the application is specified on the more abstract level of his new languages, the translation into code could proceed automatically or in a heavily machine-aided way.

Short of completely automated programming, practical tools for checking programs will result. "One of the most useful features of structured programming is that it prevents you from making certain kinds of mundane errors—the kind caught by a syntax checker, for example. At the level of my new languages, you could check for certain kinds of conceptual errors which wouldn't get caught by a compiler," explains Winograd.

An enhanced ability to reuse code modules is another important goal. Detail is rendered idiosyncratic by the language used to code the program, requiring a difficult line-by-line translation to move from one system to another. "By going up to higher levels of abstraction you get more 'shareability,'" Winograd says. "I think you can build libraries of these things."

Winograd is quick to point out that this is not a new idea. "It's the same idea that you had in FORTRAN, which is if you start at the bottom and ask 'What does the machine do?' you can gradually work your way up to things which are closer to what the problem is and away from the details of load register, store register, multiply, and so on," he explains.

Automating the programming process will shift the programmer's role. "I believe there will always be crucial decisions for programmers to make. End users will always need somebody to go in and say, 'No, don't implement that feature that way, do it this way,' or 'We should represent that feature with this sort of data structure because another structure won't be as efficient,' and the systems programmer will move to the conceptual level of saying, 'OK, we need files, we need directories, we need to deal with nodes, with networks.' Programmers are already doing that kind of thinking in an informal way," Winograd says, "because if they didn't, they wouldn't know where to start with their code. What I'm trying to do will make the process more systematic and give them more help."

Winograd predicts that systems utilities, compilers, and most of the other programming tools currently in use will be integrated into this new generation of system development languages. Depending on the requirements of a particular situation, the fundamental algorithm that structures the program may or may not need to be thought of by the programmer. "There are certain aspects where, given a declarative, nonprocedural statement of relationships, the machine could select the algorithm by itself; but that's limited to certain aspects of programming, and within those aspects I would like to see that happen. In other places what you would need to specify is precisely the algorithm, only at a higher level of abstraction," he observes.

The Ultimate Programmer's Editor
WENDIN'S XTC™

SUPER PROGRAMMERS edit in XTC to make software development a snap! Just look at these powerful features:

MULTITASKING

XTC's built-in multitasking lets you run your macros in the foreground or independently in the background while you continue editing. A background process has full access to editor resources, and can be used to translate code from one language to another in REAL TIME, print files in the background, or even scan syntax while you type in code. Best of all, you can use XTC to edit source and documentation in any programming language!

COMPILE IN WINDOWS

All DOS compilers and utilities can be executed from within XTC using a single keystroke. While it runs, XTC captures your compiler's output and redirects it into your text, so you can compare compiler messages with your source code ON THE SAME SCREEN. And using XTC's macro language, Turbo Pascal is literally only a keystroke away. You can use other compilers and utilities inside XTC too — like Lattice "C," Microsoft Pascal, and IBM's Basic, to name a few.

MACRO LANGUAGE

XTC has the most powerful macro langague in the editing world. XTC's macros aren't just keystrokes assigned to keys; they're real programs that can be used to automatically edit source code and data files. Like any real programming language, XTC has control structures like IF THEN ELSE, WHILE DO, REPEAT UNTIL, FOR NEXT, DUPLICATE N TIMES, INDEFINITE LOOP, EXIT, and BREAK LOOP. XTC also has INTEGER, BOOLEAN, and STRING variables to hold numbers, conditions, and pieces of text.

WINDOWS & BUFFERS

With XTC you can display up to 8 different files or parts of the same file on the screen at once. XTC's windows are programmable and can even be linked together to share files. XTC also has 20 other buffers that you can use to hold files and blocks of text.

INCREDIBLE EXTRAS!

XTC comes with 7,000 lines of source code jam-packed onto two DSDD disks. Includes 13 modules written in Pascal, and 2 assembly libraries you can use to access the PC's screen, intercept software interrupts (like INT 21H functions), allocate and deallocate memory, and load and execute programs. It's all included FREE for your recreation and enjoyment!

- UNDO N TIMES
- REMOVE WORDSTAR HIGH BITS
- EDIT GRAPHICS DISPLAYS
- AUTOINDENTING MODE
- TAB EXPANSION/ COMPRESSION MODE
- EXTRA LONG LINES
- MACRO COMPILER
- TELEPHONE SUPPORT
- 150 PAGES OF DOCUMENTATION
- RUNS ON IBM / PC, XT, AND / AT COMPUTERS (AND TRUE COMPATIBLES)

INCLUDED FREE!
- Macro Compiler with full source code in "C"!

To get your copy of XTC now, order it over the phone — we can ship it the same day you call! Or, send in an order, just like this one:

XTC	**$99.00**
Macro Compiler	**FREE**
Shipping, Handling, Insurance	**3.50**
Want it COD? Add this	**1.90**

WENDIN™
BOX 266
CHENEY, WA 99004
The people who make quality software tools affordable.

ORDER HOTLINE
509/235-8088
CREDIT CARDS WELCOME!

Ada® is a registered trademark of the U.S. Department of Defense.
Turbo Pascal is a trademark of Borland, Inc. XTC is a trademark of Wendin, Inc.

CIRCLE 43 ON READER SERVICE CARD

Winograd's research interest and his exploration of the larger philosophical and metaphysical implications of artificial intelligence are deeply intertwined with a finely developed sense of the social structures within which computers exist.

Winograd does not apply for or accept any Pentagon funding for his research. His view is that the military way of doing business and looking at the world has a negative impact on researchers and on society as a whole. "In addition to the world danger created by the weapons spiral, there is also a strong danger in the increasing role for the military in the U.S. in general. As an employer, a purchaser, a lobbying group, a trainer of people, and many other things, they have a negative influence on attitudes and practices," he said in a statement circulated among computer professionals in a newsletter put out by Computer Professionals for Social Responsibility, the Palo Alto, Calif.-based organization where he serves as a board member. "My view is based on wanting to decrease the overall militarization of the country."

Winograd's concern is that U.S. Dept. of Defense funding shifts the direction of research away from constructive uses and toward the development of nonproductive resources. He sees military research as a subtle legitimization of the direction the military is taking that goes far beyond the particular project being worked on. "A scientist working on parallel processing algorithms using Lawrence Berkeley Laboratory [a center for advanced nuclear weapons research based in Berkeley, Calif.] computers might be less inclined to protest what goes on there because of the dependency fostered by the relationship with DOD funding," he explains.

Involving respected university researchers in Pentagon programs enlists public support for controversial programs that otherwise might have no chance of success. "If the supporters of such a program can point to a famous scientist at a highly esteemed university working on research, it adds legitimacy and authority to the program. It also enlists support by developing a feeling of everyone being a part of the same team, working together in the direction of a shared goal," says Winograd.

He is also concerned with Pentagon restrictions on the free flow of information, attendance of professional conferences, and the classification of research findings. He worries about the next generation of students being socialized by working in programs related to military research. A student who spends four years studying high-speed electronic circuitry in weapon systems research may find it more difficult to find a job in nondefense-related areas and will consequently be channeled into working for military contractors after leaving the university. "The balance of research and teaching, of science and humanities, of graduate and undergraduate education are all shifted to suit the needs and tastes of the generals," Winograd says.

Colleagues such as AI pioneer John McCarthy challenge this view, believing that the Soviet Union is a dangerous and unpredictable adversary and that AI researchers have a patriotic duty to work with the Pentagon to counter Soviet research efforts. Winograd argues that patriotism is not the act of building more sophisticated weapons, because more weapons will not make the world safer.

"If you proceed in building more and better weapon systems, will you produce a situation where we are alive and have preserved our democratic values?" asks Winograd. "My answer is no, that this mentality is going to lead us into situations which have a greater probability of ending up with everybody dead, both us and the Soviets," he says.

Winograd advocates an evolutionary process of raising public awareness to the dangers of military funding and applications of computer research. "If somebody asked, 'Would you advocate next week getting rid of the Defense Dept.?' it wouldn't make sense. But if you ask, 'Do you think the ideal world, one that you might want to work toward, is one in which everybody agrees to abolish the Defense Dept.?' I'd say, 'Sure,'" he explains.

Winograd acknowledges the privileged position that allows him the luxury of refusing DOD money. "I'm at Stanford, in Silicon Valley, with a lot of connections; as long as there is a certain amount of money that is not military I have as good a chance as anybody to get that money. For somebody not situated as centrally, it's more of a problem. If you are the person doing research in some small place, you're not likely to get other funding, not likely to have many contacts," he says.

Each person has to decide how to handle military funding. "I don't think these kinds of moral issues have one right answer. The research can be removed far from any actual killing. If you're asked to pull the trigger, that's one thing. But if you're asked to design a computer programming language which could be used to write programs that could run in machines that kill, you know your language could just as easily be used to program hospital applications," Winograd says. "Every case is a hard case."

In evaluating the future of AI, Winograd notes that so far computer applications have largely replicated tasks already done manually, as in the replacement of the typewriter with word processing or of calculators with spreadsheet software. This is to be expected following the introduction of new technology, he believes. He observes that the appearance of the telephone did not magically produce today's work places and activities. "But revolutionary new possibilities like multinational companies and other ways of doing business that wouldn't be practical without the telephone eventually changed the nature of business," he says.

He expects a social transformation made possible by computers to start in the business sectors of our society. "The only things which are really going to succeed are those which affect what people do. Typically, that means what people do at work," he says. Using the computer to make tasks easier and more efficient is a natural response, fitting the machine into the existing network of practices and ways of working.

Farther in the future are truly revolutionary ways of doing things made possible by capabilities inherent in the computer. "I think the biggest potential for computer application is in communications," Winograd says. "Rather than thinking of the machine as doing a calculation or a computation, I think of it as one part of a network connecting satellites and telephones and data bases and so on, a network filtering things that one person wants to put on and another person wants to take off the system."

The simplest example is electronic mail, but as machines become more sophisticated, incorporating visual and other kinds of information, he foresees substantial changes. As a consultant for Action Technologies, a small San Francisco, Calif.-based software company, he is targeting just this kind of fundamental advance. "One of the things I'm working on is a program that tries to use the computer in interesting ways to restructure interpersonal communications," he says.

Reflecting broad concerns about the application of AI research, Winograd says technology must complement human activity, not replace it. "I'm skeptical of any attempt to solve complex social, political, human problems by coming up with the right technology," he says. "I'm hoping people will become aware that what computers are doing now is just one useful part of intelligence. We need to understand where, in the collection of the various things we are and all the things we do, computers can be best applied."

You know that choosing the right software is serious business. So does WATCOM.

So before you make any decisions about your software needs, talk to WATCOM—the people major software users around the world have trusted for years.

WATCOM has the products you need to get the job done right. Proven performers like WATFOR*, WATFIV*, WATBOL*, and SCRIPT. Plus new leaders in software for PC workstations and micro-to-mainframe communications. Networks, language interpreters and compilers. Text preparation and data management.

All WATCOM products are human engineered to provide the optimum in people efficiency and productivity. And they're designed to run compatibly on IBM mainframes and PC's, Digital mainframes and micros, and Commodore micros.

Whatever you need is backed up by WATCOM's innovative maintenance and support services. You'll be kept up to date with the latest in product enhancements and information. And our publications and seminars will help you get the most out of your software investment.

WATCOM. Quality products. Professional service. And a reputation built on more than 150,000 licensed mainframe and micro software programs throughout the world. So talk to us before you decide. After all, choosing the right software is serious business. For you. And for WATCOM.

Make the Right Choice: WATFOR-77 — PC and 370

Optimizing programmer productivity makes WATFOR-77 the right choice.

WATFOR-77 significantly reduces the time required for program development by compiling and executing the program directly in the computer memory in one step. Together with excellent debugging and error diagnostics, WATFOR-77 optimizes programmer efficiency and overall productivity.

WATFOR-77 is the newest member of the WATFOR family of debugging FORTRAN compilers and is ideal in business, educational, and scientific applications. Supporting the full FORTRAN 77 language, WATFOR-77 is now available for the IBM PC* DOS and 370 VM/SP CMS and MVS*.

Make the right choice. Call or write WATCOM today and we'll tell you all about WATFOR-77 or any of WATCOM's other people-efficient products.

WATCOM

The right choice in software.

Yes! I want to make the right choice in software. Send me more information on: ☐ WATFOR 77 ☐ WATCOM Software Catalogue

Name:________________________

Company:________________________

Title:________________________

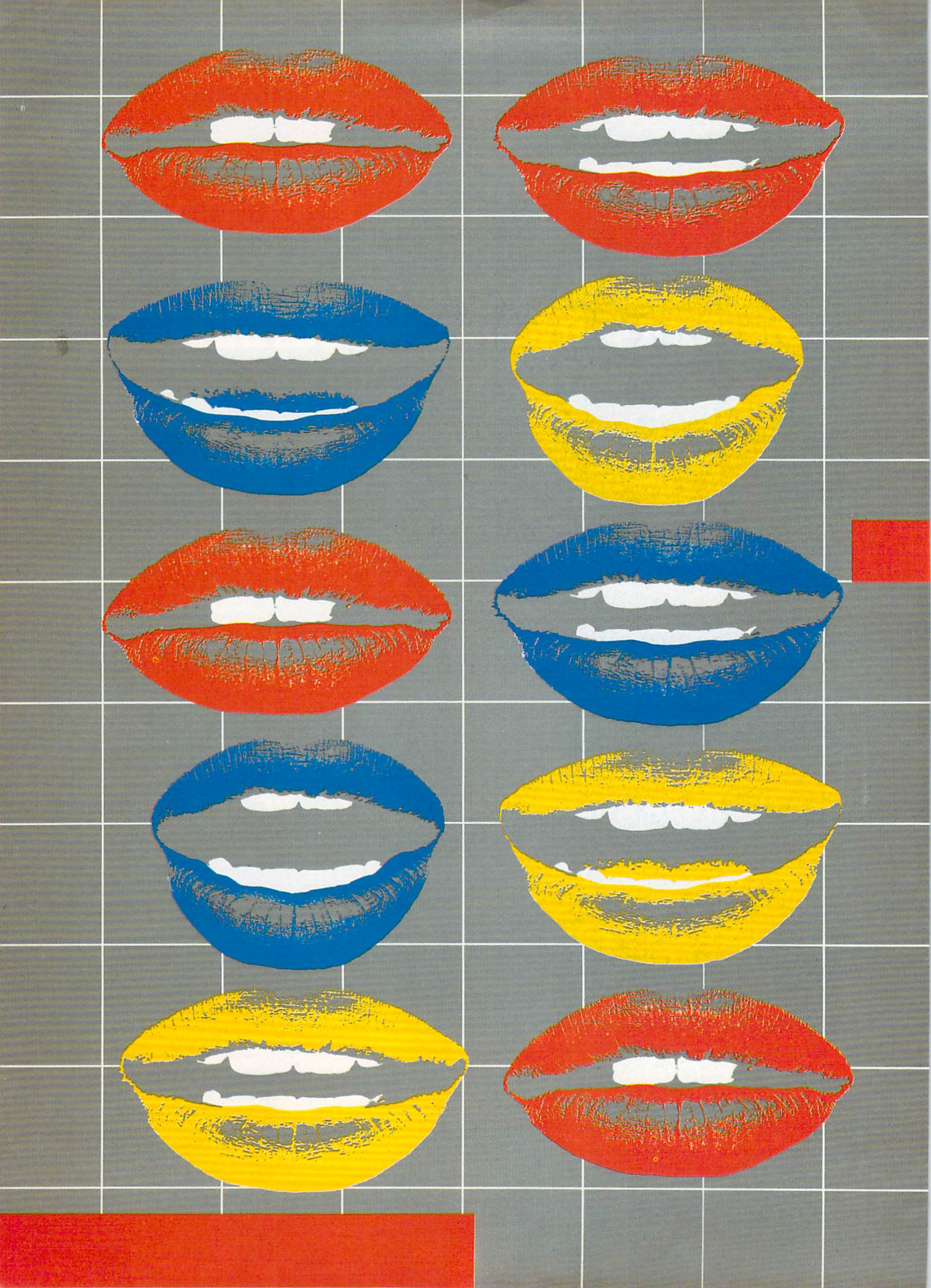
Address:________________________

City:__________ **State:**__________ **Zip:**__________

WATCOM PRODUCTS INC.
415 Phillip Street
Waterloo, Ontario, Canada
N2L 3X2
(519) 886-3700
Telex 06-955458

* WATFOR, WATFIV and WATBOL are registered trademarks of the University of Waterloo.
* IBM PC and IBM 370 VM/SP CMS and MVS are registered trademarks of International Business Machines Corporation.

CIRCLE 36 ON READER SERVICE CARD



Speaking LISP

By John R. Allen

Along with the growing interest in artificial intelligence comes an increased curiosity about the flagship language of the AI community—LISP. In this article we hope to satisfy that curiosity with a characterization of what it means for a language to be LISP-like.

The major concern people raise about LISP is its unique syntax—the parentheses that decorate LISP expressions make for hard reading. Traditional languages, like FORTRAN or Pascal or even C, are much less confusing to the human reader—or so the claim goes.

We will show that LISP's syntax is only a minor point in the class of issues that distinguishes LISP from the traditional languages. The deep issues that separate LISP are semantic notions and an outlook on the computational notations we call computer languages.

To illuminate the LISP perspective, we will make repeated use of the following problem: given a finite sequence of numbers, compute their sum.

A sequence is just that: a collection of objects, ordered by the integers. We would write a sequence as (*e1, . . . ,en*) where each *ei* is to be a number. Then, for example, the sequence (1, 2, 3) would have a sum of 6, and the sequence (3.1, 12, 8.3, 3, 22) would have a sum of 48.4.

In a few moments, we should be able to convince ourselves that we can compute the sum of any sequence of numbers. And so the next task would be to transform this innocuous problem into a specific programming language.

First, the notion of sequence is usually transformed into the concrete form of a vector. Next we could transform our informal notion of summation into a precise piece of code. For example, it could be written in FORTRAN as:

```
    X = 0
    DO 400 I = 1, LEN
      X = A(I) + X
400 CONTINUE
```

where *LEN* represents the length of the vector. Or in Pascal we could write:

```
x := 0;
for i := 1 to len do
x := a [i] + x
```

Now, a LISP version need not be much different:

```
(setq x 0)
(for (i 1 len) (setq x (+ (vref a i) x))
```

We can see that the assignment *X = 0* or *x := 0* is represented in LISP as:

```
(setq x 0)
```

This example illustrates other points about LISP.

- The general form of an expression is a list of elements, which is designated by surrounding those elements with parentheses.
- The first element in a LISP expression usually designates an operation. The exception occurs in the control constructs like *for.*

Looking at the LISP code, we see that *for, setq,* +, and *vref* represent LISP operations. (LISP examples in this article are from The LISP Co.'s TLC-LISP.)

Parentheses in LISP are used to indicate the boundaries of an operation, much like the *do-continue, do-;* or {-} constructions of FORTRAN, Pascal, or C, respectively. So at least syntactically, there's not much difference between LISP, Pascal, and FORTRAN in this example.

Now we would like to describe a function of one argument (which is a sequence), whose body will compute the sum of the elements and whose final value is that sum. Thus, in a mathematical notation we might write:

```
sum ( (1, 2, 3) ) = 6
sum ( (3.1, 12, 8.3, 3, 22) ) = 48.4
```

It would seem that the definition of *sum* would be a simple wrapper around any of our program fragments. That wrapper would say:

- Make the following code synonymous with the name sum.
- Here's a name (formal parameter) to use as a synonym for the specific vector (actual parameter) you wish to sum.

Unfortunately there's a rub between the wrapper and the program segments: how do you determine the length of the vector? Of course there are programming tricks—where there's a will, there's a kludge:

- Place some end-of-elements indicator after the last "real" element in any vector and then compute the running sum until that indicator is encountered, as in:

```
100 LET X = 0
200 LET I = 1
300 IF A(I) = 99999 GOTO 700
400 LET X = A(I) + X
500 LET I = I + 1
600 GOTO 300
700 . . .
```

■ Or always make the first element in the vector be an indication of the length of the vector:

```
type vector = array [1 . . . 34] of real;
function sumvec (a: vector): real
     var x, len: real; i:integer;
     len := a[1];
     x := 0;
     for i := 2 to len do
     x := a[i] + x;
     sumvec := x
end
```

■ And of course there's the total capitulation—always pass an explicit parameter that represents the length of the vector. Thus, in FORTRAN:

```
     FUNCTION SUMVEC (A, LEN)
     DIMENSION A(LEN)
     X = 0
     DO 100 I = 1, LEN
     X = A(I) + X
100 CONTINUE
     SUMVEC = X
     RETURN
     END
```

First-class objects

One might argue "That's not so bad. Whenever we're asked to sum up some numbers, we have to know how many we've got. We need some way to tell when we've seen the last one." But that misses the point: we want to think of the sequence as a single object. We want to settle the issue of "how many" when the sequence object is created, and we want to create different sized sequences dynamically (while our computational conversation is proceeding). After the object is created, we want to manipulate the sequence as an entity. What happens to this simple sequence-idea when it gets tied up in a programming language?

Mathematics treats sequences as abstract entities, just like it treats numbers as abstractions. There is no notion of a largest integer in the language of mathematics. In contrast, computer languages tend to constrain their numbers by the (capricious) size of a machine word.

Similarly, computer languages tend to impose implementation-dependent restrictions on sequences when they are transformed into vectors. These restrictions are the software restrictions rather than hardware constraints. Certain care is required in the design and implementation of computer languages to make vectors (and numbers) behave like their mathematical brethren.

In earlier days, we could afford to ignore such elegance of notation because elegance meant time and space, and machines were of limited memory and expensive. But the balance has shifted: machines are speedy and cheap; programmers' aspirations have grown along with their salaries. We need languages that make it easy to express the complex, and that means elegance.

The most oft-quoted declaration of elegance in computer languages comes from Robin Poplestone, a British computer scientist, who expressed the idea of "first-class items." A language that deals with items—what we'd call objects—in a first-class fashion is well on the road to elegance.

The four main tenets of first-classness are:

1. Any object may be passed as a parameter to a function
2. Any object may be returned as a value from a function
3. Any object may be assigned as a value to an identifier
4. Any two objects may be tested for equality.

Most traditional languages do an abysmal job with objects as vanilla as vectors, frequently prohibiting vectors as parameters and almost always disallowing vectors as return values. The major exception is APL. In that language, arrays are treated with respect. So too in LISP, where we strive for the ideal of first-class objects in a more general setting.

If we want vectors to be first-class objects, then functions that receive vector objects as arguments must be able to determine the size of the vector.

So given a vector object we assume we have a function named *length* that will compute its size. Then we could even write in pseudo FORTRAN:

```
     FUNCTION SUM (A)
     LEN = LENGTH (A)
     X = 0
     DO 400 I = 1, LEN
     X = A(I) + X
400 CONTINUE
     RETURN (X)
     END
```

or in psuedo C:

```
sum (a)
     int a [];
     { int i;
          int x = 0;
          for (i=0; length (s); ++i)
               x = a [i] + x;
          return (x); }
```

or in LISP:

```
(de sum (v &aux (x 0))
     (for (i 1 (length v))
               (setq x (add x (vref v i))))
     x))
```

Even without a formal discussion of syntax, it's reasonably easy to understand each dialect of the summation program. The important point here is the appearance of the *return* in C and FORTRAN and its absence in the LISP code. The issue is elegance again; this time the issue involves the treatment of program modules rather than data objects.

Functional languages

By way of background, most languages (Pascal, FORTRAN, BASIC, C, and traditional Logo, for example) are built on a von Neumann model of computation. By this we mean that the computational engine is thought of as a collection of cells, each able to contain one piece of information. The actions of the machine

consist of selecting the contents of some cells, performing an operation on those contents, and replacing the contents of a cell with the results.

The inherent characteristic is called a side-effect, meaning that the intent of computation is to change the contents of a cell. In a higher level language (like C, FORTRAN, and Pascal), we find this model of computation represented by linguistic constructs called statements. These traditional languages also include constructs to collect and parameterize computations. A collection of statements is called a procedure. We therefore call such languages procedural languages.

An alternative to the side-effect model is the value model. Here a computation is envisioned as producing a value as in "the value of 3+4 is 7." Compare that with: "to compute 3+4, place 3 in a cell, place 4 in a cell, operate on these two cells with the plus operation, and then place the result in another cell." In the value model, one is more concerned with a description of a computation rather than a specific prescription of how to perform that computation.

Value-based languages have a construct comparable to the statement in the procedural languages. This construct, called an expression, always produces a value. This regularity of value-based languages extends to the constructs that build new computational units. These units are called functions and are based on the mathematical notion of function. As a result, these languages are called functional languages. Sometimes they are referred to as applicative languages, because the general way to compute new values is to apply functions to arguments. Such languages include LISP, Scheme, and TLC-Logo.

The problem here is again semantic as well as syntactic. Namely, the requirement that functions be notated to return values spoils the elegance of the mathematical base, making functions second-class to procedures. The benefits of a functional language over a procedural one are more than cosmetic. The flexibility of computational models for a functional language is comparable to its notational flexibility. Specifically, in functional languages complex computations are expressed by functional composition as in:

```
f(g(x, y), h(z))
```

In mathematics, such an expression simply denotes a value— there is no sense of computation. In a functional programming language like (pure) LISP we'd write:

```
(F (G X Y) (H Z))
```

and would expect most LISP engines to compute *(G X Y)*, then *(H Z)*, and finally pass those two results to *F*. That's purely a historical prejudice: if we had several LISP engines, the *F*-engine could request two other engines to begin the simultaneous execution of *(G X Y)* and *(H Z)*, respectively. We might expect that the final computation would be completed faster than its sequential cousin:

```
(SETQ T1 (G X Y))
(SETQ T2 (H Z))
(F T1 T2)
```

The functional expression of the problem is much more elegant and understandable. And isn't the point of notation to clarify a situation?

A further benefit to the functional approach is its clear and clean model of computation. Two kinds of rules—substitution and simplification—are all that's required to characterize functional computation. For example, given:

```
(de example (xy)
    (if(gt x y) (+ x 4)
           (- x 3)))
```

we could compute *(example 12 5)* by substituting 12 for *x* and 5 for *y* throughout the definition of example:

```
(if (gt 12 5) (+ 12 4)
    (- 12 3)
```

We could simplify *(gt 12 5)* getting *T*, reducing this expression to:

```
(if T (+ 12 4)
    (- 12 3)
```

Now we could invoke a rule *(if T <anything1> <anything2>)* reduces to *<anything1>* and get *(+ 12 4)*, and finally simplify that to 16.

First-class functions

The appropriate definition and application of such rules are at the heart of computation, and the accurate implementation of these rules is the heart of computer science. For example, we don't usually make explicit substitutions in the body of a function; rather we simulate such operations using a symbol table to contain the associations between names and values.

Though the proper implementation of such rules is difficult, the underlying model is far more rational than that offered by the state-change/side-effect world of procedural languages. To a large extent, computation is just applied logic; the logic of functional languages is substantially cleaner than the logic of procedural ones.

Of course, these are arguments for and against a purely functional and a purely procedural language. Real languages are never so pure; what we're after in our characterization is intention. The intent of a LISP-like language is functional. The intent of a traditional language is procedural.

So we've captured two characteristics of a LISP-like language: a treatment of data objects in a first-class fashion and a computational notation based on function. Such a language is still pretty traditional: objects are first-class but are just sanitized versions of the data we've seen in other languages, and the programming language is functional but is just a simplified version of the functional portions of a half-dozen extant languages.

But what happens if functions are allowed to become objects of the language and (of course) granted first-class status? This question has two portions: what practical benefit (notationally or computationally) does such a mixing of program and data give us, and assuming that those benefits are substantial, what happens to our model of implementation?

Let's go back to our vector example. Let's assume that we want to build an APL-like language inside our LISP-like language. We'll need operations on vectors to create new vectors. Let's start easy. Assume we wish to construct a new vector each of whose elements is twice the original element. So if we call that function *double*, then:

```
(double [1 2 3]) = = > [2 4 6]
```

or

```
(double [3.1 12 8.3 3 22])
    = = > [6.2 24 16.6 6 44]
```

and of course functional composition is available to us:

```
(sum (double [1 2 3])) = = > 12
```

The elegance of the result is apparent. Its execution had better be equally elegant in that *double* must put its result somewhere and it cannot destroy the input argument. What about the LISP code to produce *double*? Behold Listing 1!

Three semantic points appear here:

1. The local creation of a vector object using *newvector*
2. The naming of that object as *v1*
3. The return of *v1* as the value of *double*.

The latter two points come from our insistence on first-class objects. The first point, the dynamic creation of new objects, is a key ingredient to making first-class objects a success and is indeed LISP-ish. This ability to create new objects out of the ether was championed by LISP's *cons* operation. Its implementation raises the specter of dynamic storage management and garbage collection and raises the hackles of those who still believe that computers are supposed to get answers rather than help formulate solutions.

So even before we get to the question of first-class functions, an insistence on first-class data pays handsome benefits. Let's continue.

Assume that, instead of simply doubling the elements of our vector, we'd like to perform some as yet undetermined operation on each element of the vector (Listing 2).

The general structure of the new function is identical to *double*. One difference appears in the calling sequence: *op* is a parameter that represents a function rather than data.

The other difference between *map-op* and *double* is: *op* is applied to the vector reference, where + appeared.

This is another of the first-class tenets—naming a function. But what is the difference between a functional object and its name? For example, we might use *map-op* to define *double* by:

```
(de double (v) (map-op twice v))
```

where

```
(de twice (x) (+ x x))
```

In this case, we have named a function twice, but the real meat of the function is: given a number, double it. We could define the same function by:

```
(de time-of-day (x) (+ x x))
```

At least in this case, then, the name of the function is irrelevant; we just need its effect. We can do this in LISP by writing:

```
(lambda (x) (+ x x))
```

or even

```
(lambda (y123) (+ y123 y123))
```

which says we also don't care what we name the number either. Regardless, we can use the lambda construct to name a *twice* function and use it internally to double:

```
(de double (v &aux (twice (lambda (x)
                                  (+ x x)))
            (map-op twice v))
```

We can *de* even better since we don't need the *twice* name:

```
(de double (v) (map-op (lambda (x)(+
   x x)) v))
```

So the lambda expression *(lambda . . .)* plays the role of a functional constant just like 2 is a numeric constant. We are used to writing:

```
2 + 2 + 3
```

rather than the more stilted:

```
for (x := 2 y := 3) compute x + x + y
```

Lambda expressions give us a similar visual advantage with functional objects. And to complete the analogy with assignments, it is appropriate to think of the structure:

```
(de <name> <parameters>
    <body>)
```

as an abbreviation for:

```
(setq <name> (lambda
      <parameters> <body>))
```

So it appears that we can separate a function from its name, and the question becomes: what is a functional object? This brings the question of scoping into play.

The term scope refers to the relationship between names (like *map-op*, *x*, and *twice*) and their values (a globally defined function, a formal parameter, and a locally defined function, respectively). The question of scope occurs because of our decision to simulate the substitution

```
(de double (v &aux (len (length v))
                   (v1 (newvector len 0)))
  (for (i 1 1) (store v1 i (+ (vref v i)
                              (vref v i))) )
  v1))
```

Listing 1.

model by symbol tables rather than make the explicit substitutions.

The appropriate model to understand scope is to think of a symbol table as a dictionary. When someone writes a phrase, they have specific meanings in mind—say they use *Webster's* dictionary. A reader of that passage who used *Funk & Wagnalls'* might get a quite different meaning for the passage. The issue of scoping is the maintenance of dictionaries so that the reader is assured access to the writer's dictionary. As we will show, maintenance of such dictionary associations requires care.

To begin, assume we simply wish to make a new vector, each of whose elements is 1 greater than the elements of the argument vector. We could do this by:

```
(map-op (lambda (x) (add 1 x))
             [1 2 3])
==> [2 3 4]
```

But suppose now that we wished to exploit a general *add-by-n* operation on vectors:

```
(de add-by-n (n v) (map-op
     (lambda (x) (add n x)) v))
```

then we'd be able to increment each element of a vector by 3 with:

```
(add-by-n 3 [1 2 3]) ==> [4 5 6]
```

For example, our substitution model will begin with:

```
(map-op (lambda (x) (add 3 x) [1 2 3])
```

and the rest is easy.

However, a slight syntactic change destroys these pretty results in a traditional LISP. If we change *add-by-n* to:

```
(de add-by-n (i v)
     (map-op (lambda (add i x)) v)),
```

replacing every occurrence of *n* by *i*, then:

```
(add-by-n 3 [1 2 3]) ==> [2 4 6] !!
```

The substitution model doesn't break down, its simulation in traditional LISP is at fault. To see this, consider the expression:

```
(op (vref v i)) inside map-op.
```

When we try to evaluate *(add-by-n 3 [1 2 3])* in LISP we'll find *op* to be associated with *(lambda (x) (add i x))*. We'll want the *i* in *(vref v i)* to be interpreted in the dictionary of *map-op*, but the *i* in *(lambda (x)(add i x))* to be interpreted in the dictionary of *add-by-n*. Traditional LISP will use the dictionary of the reader *(map-op)* for both occurrences of *i* and will get the wrong answer.

The dictionary that is present when a function is defined (or written) contains the lexical or static binding of a symbol. The dictionary employed by the reader of a phrase contains the dynamic or latest binding. The distinction can be ignored without peril until a language begins to tinker with a functional object. Then it makes a difference.

In situations like the preceding one, dynamic binding is patently a bug. In other situations—particularly in interactive settings—the case is not that clear cut. For example, if we have a program that contains expressions involving output:

```
.... (print x) ....
```

we might decide that we'd rather direct the output into a vector in memory. If the name *print* is dynamically bound, then we could accomplish this by:

```
((lambda (print) ... (print x) ... )
     <unary function to stuff vector>)
```

whereas if *print* were lexically scoped then the name would have already been bound to the system-level printer.

The issue of scoping is a reasonably deep one; its pitfalls and initial resolution in a fully functional setting were first discovered in LISP implementations. The general solution is notationally straightforward—substitution of actual values for formal parameters.

However, such an implementation is expensive and would ignore some powerful computational models of state-oriented computation. As a result, most LISPs that address the problem use a device called a closure to capture variable bindings (like the illusive 3 associated with *i*). A closure object consists of a piece of function text (the lambda construct) coupled with a symbol table (also called an environment or dictionary), and the table contains information about bindings that should be installed when a closure is applied. Thus in TLC-LISP we write the code in Listing 3, and the right thing will happen.

Interesting. We've gone through some of the deepest areas that make LISP LISP, and we've not even talked about list pro-

```
(de map-op (op v &aux (len (length v))
                      (v1 (newvector len 0)))
  (for (i 1 1) (store v1 i (op (vref v i))
  v1))
```

Listing 2.

```
(de add-by-n (i v)
 (map-op  (closure (lambda (x) (add i x)) (env 'i i))
          v))
```

Listing 3.

cessing. That was intentional: the issues of functional languages and first-class objects transcend any concern for specific data objects—even LISP-like lists.

Lists and vectors are both implementations of the notion of a finite sequence. Traditionally, vectors have been used to represent objects whose general structure is fixed so that the successor/predecessor relationships become +1/−1. Lists are more appropriate in applications that involve highly variable structures; topological relationships may change, grow, and shrink in unpredictable fashions. In particular, vectors usually come into existence as a unit, while a list will be built up in an element-by-element fashion.

The issue of notation for abstract objects in programming languages is not just a theoretical diversion. As we've just seen, our ability to determine characteristics of objects dynamically in LISP gives freedom to our modes of expression.

LISP as a starting point

Rather than dwell on the different types of data in a modern LISP, or trot out a suite of LISP applications in expert systems, we'd rather indicate a few directions that can be explored from our new vantage point.

- Toward simplicity of notation
- Toward generality of expression
- Toward deep applications.

Toward simplicity: Logo. Though the dominant characteristic of Logo is its turtle graphics, Logo's syntactic characteristics are a direct result of simplifying LISP's syntax.

In the late 1960s a group at Bolt, Beranek, and Neuman in Cambridge, Mass., recognized that the power and flexibility of programming style enjoyed by the LISP community was transferrable to domains other than artificial intelligence. In particular, LISP was highly interactive and encouraged an exploratory and experimental style of program development.

Their challenge was to disentangle LISP's power from the specifics of its syntax. Since LISP expressions are written in prefix form, one could dispense with the parentheses if the parity of each function were known. Thus if *F* were a function of three arguments, then *(F 1 2 3)* could be represented as *F 1 2 3.*

Even functional composition could be accommodated. So if *G* were a binary function, then *(F (G 2 3) 4 5)* could be represented as *F G 2 3 4 5.*

But what about *(F 1 2 X)*? We couldn't write it as *F 1 2 X* because that would represent *(F 1 2 (X)).* So an additional artifact called dots (:) was invented to prefix occurrences of symbols that were to represent variables. *(F 1 2 X)* would be represented as *F 1 2 :X.*

With a few other syntactic embellishments, the Logo group tamed LISP's syntax. Unfortunately, they also took liberties with LISP's semantics, making the default module a procedure and requiring additional syntax to make a function—a procedure that returns a value. This semantic change makes it difficult to justify the often heard comment that "Logo is a LISP-like language." Regardless, to the novice at least, traditional Logo looks like a baby LISP without parentheses: data are numbers, symbols, and lists, but it has no vectors or functional objects.

But it is now 15 years later. People's expectations and experience with languages have increased. As a result we believe it is important to return to the LISP roots when designing Logo-like languages.

Toward expressivity: logic programming. We can move another direction from the LISP base, generalizing from the functional programming base to the relational programming languages.

The theoretical work that supports the functional programming languages is a product of research into the foundations of mathematics. Specifically, a mathematical formalism called lambda calculus was invented by Alonzo Church in the 1930-1940 time frame. That language gave axioms and rules of inference that characterized the mathematical notions of function.

In that same time period, formal systems were invented to give precision to the notion of relation. A mathematical relation is a generalization of the notion of function. As such, programming languages based on relations are potentially more general than their functional cousins. The specific formal system that supports these relational or logical languages is called first-order predicate calculus.

We will not go into specifics of logic programming here. The point to be made is that the commonality of mathematical pedigree distinguishes the functional and relational languages from their procedural cousins. The style of thinking, the attention to abstraction, and the importance of notation over specifics of implementation—these are common features of the functional and relational languages.

With the relational languages, we now have the basis for a comparison of the components in the currently available programming notations (Table 1).

To some extent the functional languages represent a compromise: they are not as general as the relational languages and not as machine-oriented as the traditional ones. With guidance, novices can map their experience in mathematics into a functional setting; they can impose a specific process regime on their model of function evaluation and functional composition. From a practical perspective, there has been a decade more experience with functional languages than with their relational compatriots. This experience

Language type

	Procedural	Functional	Relational
Module	procedure	function	clause
Component	statement	expression	relation
Naming	assignment	lambda binding	unification
Result	side-effect	value	constraint

Table 1.

involves both the design of large systems as well as the design of the languages themselves.

Toward deep applications: introspection. No paper on LISP would be complete without a discussion of LISP's ability to convert data into programs and programs into data.

First, notice this notion is distinct from allowing functional objects—even first-class functional objects. Nothing we have done prior allows us to do any more than create functional constants, pass them as parameters, give them names, or return functions as values. We're asking for something more here: the ability to dissect, modify, and rebuild components of functions and expressions and then execute the resulting structure.

The usual reaction to such shenanigans is something akin to what Dr. Frankenstein must have heard. However, there is nothing sacrilegious about programs that can modify other programs. In a trivial sense, text editors are used this way; in a deep sense, machine-level debuggers do this too. LISP simply dignifies and cleanses the process a bit by moving the idea up to the level of a functional notation.

LISP's technique involves representing program elements as data objects—lists, in particular. As a result, programs look like data and can be manipulated by LISP's list operations. To complete the circle, we need an operation to take a data object and evaluate it; that operation is named *eval*.

There is another piece of the LISP puzzle that is related to, but not identical to, the program-as-data issue: control-as-data. The program portion of a computation represents the static information—the text and data of the process.

In the course of the computation the execution device must maintain some internal, dynamic, information—where to return to after a computation is completed, where to place values, and so on. This kind of information is called control. If we make such information explicit—make control a first-class data object—then new insights on computation become possible. In particular, if the control structures of the engine that is performing a computation are made available to that engine, that gives us the rudiments of computational introspection, which is an interesting topic of current research.

The LISP elephant

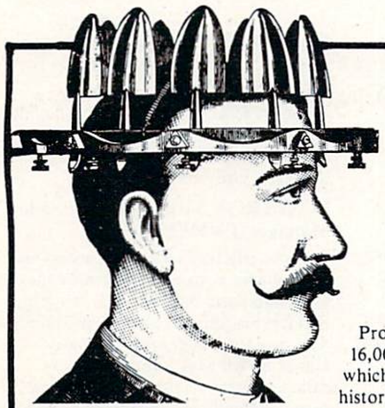
Well where are we? No useless "here's CAR and here's CDR." No insulting examples of ELIZA or ANIMAL dressed up as expert systems. No tedious "here's factorial and here's how to reverse a list."

I hope that our view of LISP is sufficiently different from what you've seen that you may be enticed to take another look. If you look at LISP with the following five notions in mind:

- First-class objects
- Functional semantics
- Functions as objects
- Programs as data
- Control as data

you'll see that the world of LISP takes on a whole new perspective. ■

John Allen is president of The LISP Company. He has been involved in LISP and AI since 1965 and is author of Anatomy of LISP *and co-author of* Thinking about TLC-Logo.



KDS gives you a better way to make powerful Expert Systems!

Produces automatically up to 16,000 rules per Knowledge Module which are induced from case histories entered using plain English.

Up to 256,000 facts per Knowledge module, for instance 4096 cases/answers and 64 attributes or 512 attributes and 512 cases/answers.

Knowledge Modules can be chained to give virtually unlimited capacity.

Can make an expert system using only plain English or/and use additional programmable features to interface to other programs such as driving external modules, get real-time input from external sensors, interface with graphics, mainframes, databases, do run-time number crunching and more.

KDS system itself written entirely in assembly language. Lightning fast!

End user can respond "?" if they don't know an answer. Powerful inference engine on playback can infer correct conclusions.

Requires only IBM-PC or compatible for development or playback. A full expert system in finished form can fit on a 5¼" floppy disk. Hard disk helpful but not required for development. PC-DOS or MS-DOS.

Forward or reverse chaining selectable. Full "explain" feature shows user how conclusions reached. Confidence factors and help screens.

Prices: Development System alone $795.00. We recommend also Playback Utility with it at $150.00. Playback Module (not required for development if you have a Playback Utility) $495.00. We welcome phone inquiries!

KDS Corporation
934 Hunter Road • Wilmette, IL 60091 • (312) 251-2621

CIRCLE 62 ON READER SERVICE CARD

tlc-lisp86

— Immediate Delivery —

Zetalisp™-like capabilities PLUS 8087 support
Smalltalk™-like class system PLUS first class instances, closures and environments
WordStar®-like display editor PLUS Lisp specific operations

ADDITIONAL FEATURES:

SOURCE FILES for Editor, Debugger and OS interface included for user customization
COMPILER for best performance of user-defined functions
CP/M, MS-DOS, PC-DOS support, Software Interrupts, full access to Memory and I/O Ports
DOCUMENTATION comprehensive Primer and Reference Manual
TLC™-Logo's MULTI-TURTLE GRAPHICS for IBM PCs

☐ **Complete System $250.**
☐ **Manual Only $20.**

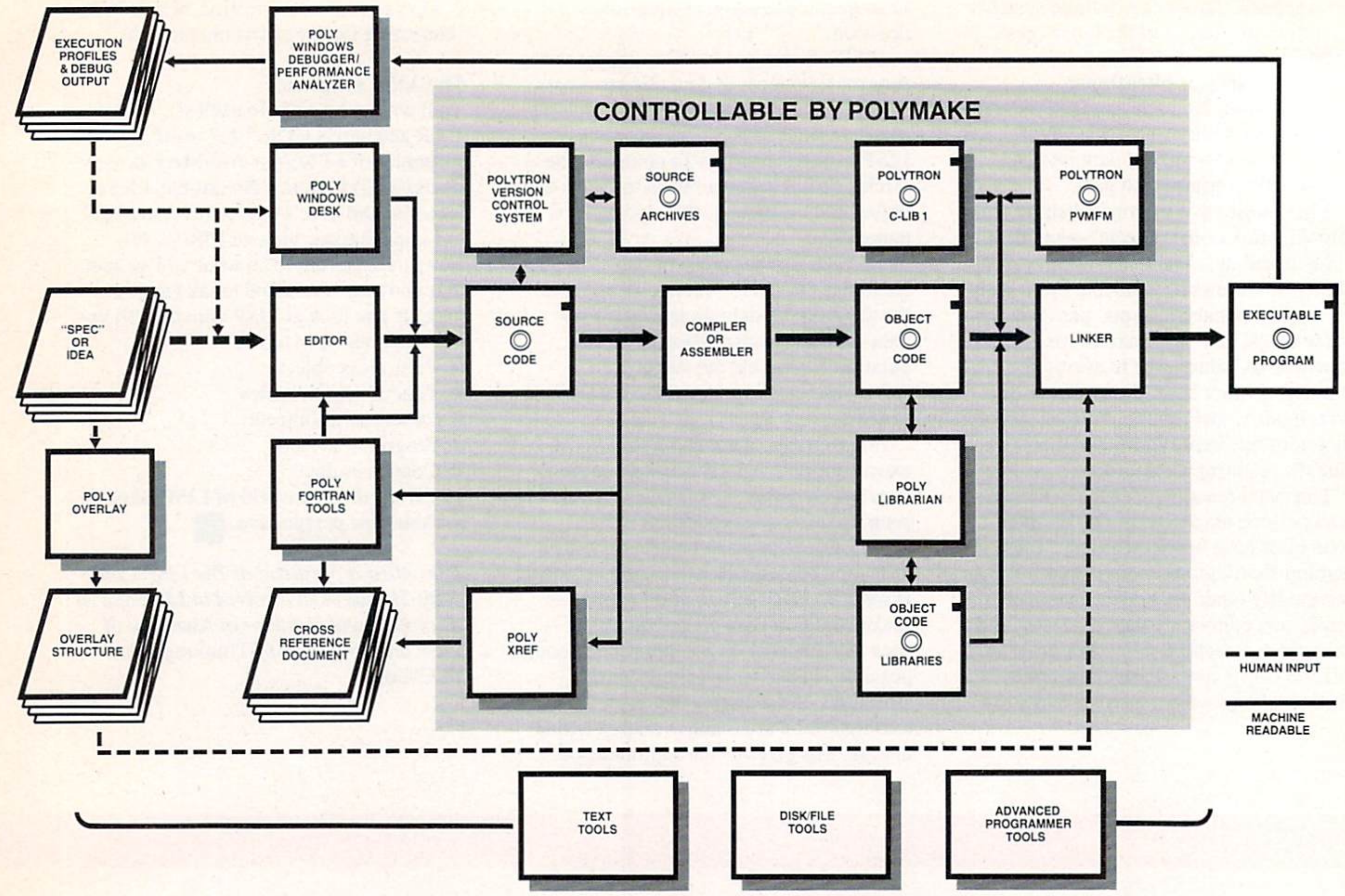
THE lisp™ COMPANY (T.(L.C))

The LISP Company (T.(L.C))
430 Monterey Ave. #4
Los Gatos, California 95030
408/354-3668

Also Available: TLC™-Logo $100., and our book Thinking About TLC™ Logo $18.
Zetalisp is a trademark of Symbolics, Inc., WordStar is a registered trademark of MicroPro International Corp., Smalltalk is a trademark of Xerox Corp.

CIRCLE 65 ON READER SERVICE CARD

POLYTRON TOOLS IN THE SOFTWARE DEVELOPMENT PROCESS



POLYTRON TOOL KITS

Thousands of Professional Software Developers with demanding deadlines turn to high performance MS-DOS/PC-DOS tools from POLYTRON to boost their productivity. All the tools shown above are explained in detail in the **POLYTRON Programmer's Catalog.**

Network & Site Licenses Available.

PolyWindows Desk
The Modular, Expandable Desktop Organizer
100% memory-resident desktop organizer for professionals. Configurable so you can have one or more of the following functions available instantly: Hex/Decimal/Binary Calculator, Standard Calculator, ASCII Table, Multiple Document Text Editor, PolyKey for creating your own keyboard macros, Rolodex-type files, Calendar, Alarm Clock, Appointment Books, Autodialer, Grabber for cut-and-paste between applications. **Copy-Protected $49.95, Not Copy-Protected $84.95.**

PolyWindows Debugger/ Performance Analyzer
A Professional, Memory-Resident Software Development Tool
Instantly analyze and debug ANY program running on your PC anytime. Superior to the IBM Professional Debugger and easier to use. Supports multiple breakpoints, multiple display formats, memory searches, single-step execution, disassembly, hex arithmetic, and block moves. Tells you where your program is spending time and allows you to optimize performance. You can specify the memory ranges for analysis. Results are displayed graphically or in tables. **Includes PolyWindows Desk (Not Copy-Protected) all for only $149.** (Present PolyWindows users can receive trade-in discounts.)

POLYTRON Version Control System (PVCS)
Efficiently maintains revision history of source files for software project management. Maintains chronological, historical records of changes as "edit scripts" (reverse deltas) with full text of the latest "checked in" revision. Reconstructs any prior "revision" of any module, defines a "version" as specified revisions of various modules, supports branching from prior revisions, optional password protection. Friendly, unobtrusive user interface is based on a unique "syntax guide" concept with context-sensitive help. Designed for single or multiple programmer projects. A powerful tool with mainframe power. **Single-user license: $395.**

PolyXREF
The Multi-Lingual Cross Reference System
Generates Cross Reference listings across single or multiple source files of one or more languages. Tells you exactly where each variable and procedure is defined and everywhere it is referenced. **Package includes PolyXREF engine plus one source language module plus .CRF module: $129, additional source language modules [C, Pascal, Assembler or English]: $49 each. Complete package including all four language modules: $219.**

PolyLibrarian
The Object Module Library Manager
The premier library manager allows you to create, examine and manage libraries. Modules can be listed, added, replaced, extracted, and deleted. You can search for public or external names and even change them. Three operating modes can be intermixed for maximum flexibility. **PolyLibrarian I for Microsoft format libraries: $99, PolyLibrarian II for Microsoft and Intel format libraries: $149.**

PolyMake
The Intelligent Program Builder and Maintenance Tool
Automates the software maintenance process. Determines which files (programs, etc.) are out of date and invokes your compiler, linker, librarian and does whatever is necessary to bring your entire system up to date. Frees you from the need to remember which files depend on others and which files have been modified. Remembers the exact sequence of operations necessary to create a new revision. Completely automatic unlike competing products. Once you use it, you can't live without it. **Only $99.**

POLYTRON C Library 1
Over 65 High Performance Routines For Lattice C Compiler Users
These are routines for serious programmers that need Executive and I/O functions. Complete source code (mostly Assembler) included. **Only $99.**

POLYTRON Virtual Memory File Manager (PVMFM)
Provides efficient virtual file access and buffering for both text and binary files with automatic buffer-to-disk swapping. Reduces program access time and handles data structures larger than memory allows. PVMFM is a library of user-callable functions that may be linked with your applications code to provide virtual file management capability and random access file buffering. Compatible with Assembler and high-level languages. **Only $199.**

To Order Products Call
1-800-547-4000
Ask for Dept. No. 350
Foreign & Oregon orders call
(503) 684-3000
Send Checks, P.O.s To:
POLYTRON Corporation
P.O. Box 787 DS-350
Hillsboro, OR 97123
Add $5.00 Shipping To Total Order

To Order detailed Programmer's Catalog, send requests to address above for fast response (or circle reader service card number).

POLYTRON
High Quality Software Since 1982

CIRCLE 34 ON READER SERVICE CARD

Logic at a Glance

Part I: The ERGO Logic Kit

By Jim McCarthy

With this article, COMPUTER LANGUAGE *launches a special three-part series that will explore the relevance of decision tables to interactive computing. The author will describe the requirements for and the design of an integrated software kit that includes a decision table editor, interpreter, compiler, grid chart editor and translator, and numerous other little and large pieces that will be worth your while.*

Since seeing something is better than reading about it, COMPUTER LANGUAGE *will also be distributing sufficient code to the ERGO Logic Kit so that you may experiment with it. (For information on how to acquire your free version of the ERGO Logic Kit, see Editor's Notes, page 5.) You will then be able to gain experience in using decision logic tables for software design, prototyping and testing, knowledge formulation and representation, and expert systems design and execution.*

There is little real doubt that computer programs will be more intelligent in the future. I do not mean our programs will be more intelligently constructed (although that is possible), but that, once constructed, they will do tasks that require more intelligence.

Today we are already seeing the first commercial realizations of the advanced programming techniques and languages often referred to collectively as artificial intelligence. The real issues are how quickly and to what extent this migration of intelligence from humans to machines will take place.

Both the rate at which intelligence finds its way into our programs and the degree of intelligence thereby proliferated are governed by two major elements.

- Translating natural intelligence into a computer friendly format requires talented programmers.
- The natural intelligence we wish to congeal in our software resides largely in the minds and experience of diverse experts who, devoting themselves to other endeavors, have little computer savvy and are probably not even fully conscious of the very thought processes that make them experts.

The computer shows no mercy toward the imprecise and diffuse intelligence of our expert. Our expert, while fully understanding say, chickens or abnormal psychology, has neither use nor concern for the niceties of computer dialectics. To engender a symbiosis between computers and experts, we must find them a common ground, a language they can both understand.

The more experts we have happily banging away at computer keyboards—exploring, encountering, expressing, encoding and archiving their wisdom—the more quickly and massively will we all have common access to our collective experience.

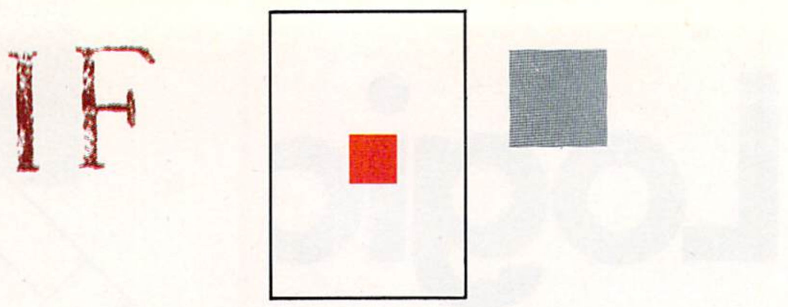
The real job, then, is not to tap experts and laboriously codify their knowledge on a case-by-case basis, with economics the sole dictator of precedence, but to provide the experts with the tools and the personal incentive to do it themselves.

If our tools cannot attain a technological state wherein experts themselves program expert systems, large numbers of such systems will not soon proliferate. Just as spreadsheets were made into a simplified form of decision support, our expectations of what artificial intelligence means may have to be scaled down to simplify the programming process and thereby greatly extend access to the technology.

And just as we were surprised by the inexhaustible creative lode that was tapped with the medium of electronic spreadsheets, we may be similarly surprised if we make available a comparably friendly medium in the field of expert systems. We won't really know what's wanted until the experts get started.

I mentioned earlier that the thought processes that constitute expertise are often obscure even to the person who has them. It has been my experience that when experts describe their methods, they usually leave out myriad hidden assumptions, exceptions to the general rule, and essential refinements to the methods described. The storage and management of a large part of their wisdom has seemingly been delegated to long-term memory and some sort of background thought process.

With the greater experts, virtually the entire matter belongs to the still more impenetrable realm of the unconscious. Intuition and something called judgement are often the primary ingredients of expertise. Lacking a straightforward



methodology and a simple format for the exploration and codification of expertise, the expert has only limited access to his or her own judgmental powers.

I believe that the experts' urge to comprehend themselves, express themselves, and extend the reach in time and space of their expertise will be the predominant incentive behind the fabrication of expert systems.

In those cases where the experts are not the programmers of the expert systems to which they are related, the ability of large numbers of programmers and analysts to fabricate logically complex software with a greater degree of confidence than is now the case is a prerequisite. The increasing complexity and consequent unreliability of software is (or soon will be) a major technological issue. The virtual impossibility of testing software threatens to swamp our productive capacity.

The purpose of this series of articles is to show one way we might make progress toward accomplishing these various ends and to make a concrete contribution in the form of a design and some source code that may stir up some interest, fuel some imaginations, or otherwise be fruitful.

I will describe a software kit[1] based upon decision logic tables. Decision tables do for certain kinds of logic what spreadsheets do for certain kinds of numbers. An electronic decision table provides many of the same qualitative experiences to the user that electronic spreadsheets do: a visceral sense of control, even mastery; the feeling of instantaneity, of certain and rapid-fire accuracy; the sheer pleasure and multiple communicative benefits of intense visual feedback; and the joy of liberating latent programming skills.

Think back to the first time you saw Visicalc, to the excitement it may have generated for you. Perhaps you even experienced a revolution in your way of thinking about computers. The idea of interactive decision tables is of a comparable magnitude: something that absolutely fits both people and machines, something that can bind the computer in a most direct way to your imagination, without a lot of folderol in between. Decision logic tables, like spreadsheets and word processing, are the type of computer application that seems perfectly obvious after you've seen it.

Decision table structure

First we must cover some fundamentals, and rather concisely at that. A decision table is a map, representing the relations of combinations of conditions to combinations of actions. A single condition/action relation might be something like:

If it's raining (condition),
wear your overcoat (action).

When multiple conditions and actions are present, things get a little more complicated:

```
If it's raining (condition1),
and cold (condition2),
and it's not expected to improve
      (condition3),
then don't go out (action1);
but if it's raining (condition1),
and not cold (condition2),
or cold (condition2)
and not raining (condition1),
then wear your overcoat (action2);
but if it's expected to get better
      (condition3)
wait until it does (action3).
```

Let's use decision table techniques to analyze this trivial problem. This is accomplished by isolating the conditions and the actions and mapping them into a tabular format divided into four quadrants:

condition stub	condition entry
action stub	action entry

Our three conditions are:

Raining (y/n)?
Cold (y/n)?
Expected to improve (y/n)?

Our three actions are:

Don't go out
Wear coat
Wait

therefore:

Raining?
Cold?
Improving?
Don't go out
Wear coat
Wait

Now we simply fill in the rules in the condition and action entry stubs of the table, like so:

Raining?	y	y	n	
Cold?	y	n	y	
Improving?	n			y
Don't go out	x			
Wear coat	x	x		
Wait				x

A careful comparison of the tabular and the textual descriptions of our little condition/action matrix will reveal that the table is merely the transcription of the problem as described.

Of significant interest here is that you likely understood the way the preceding table was to be read. Note that I said nothing of the actual mechanics of communication in the structure of a decision table; rather, your intuition grasped (probably) the meaning due to the orderly, graphical layout of the information content of the problem. If the only benefit of decision tables were their communicative properties, that would probably be sufficient cause for their use. However, there is much more value to the structure, as we shall see.

Matrix, column counts, don't cares

In the preceding table, we had some blank condition entries (for example, second column, entry for Improving?). We will fill these blanks in with dashes (—), and we will understand these dashes to mean "don't care." Also, let's add a matrix column (M) so we can do some useful arithmetic. Finally, let's add some labels to

columns (from now on called rules) and rows so that we can refer to them conveniently. Our table now looks like this:

	M	1	2	3	4
C1 Raining?	2	y	y	n	—
C2 Cold?	2	y	n	y	—
C3 Improving?	2	n	—	—	y
A1 Don't go out		x			
A2 Wear Coat			x	x	
A3 Wait					x

The M column designates the number of values a condition can attain. In our case, all the conditions are a simple Yes/No, so 2 is our uniform entry in the matrix column.

A decision table made up solely of conditions that have a binary (or y/n) matrix is called a limited entry table. Tables having only conditions that have greater than a binary matrix are called extended entry tables. Finally, tables that have both binary and greater than binary condition matrices are called, logically enough, mixed entry tables. We will give examples of mixed tables later on.

The virtue of including the M column is that now we can determine how many rules should be in the table. If we have three conditions, each of which can attain two values, the total decision matrix should be 2*2*2, or 2^3, or 8. Consider: if condition C1 can attain two states, and likewise with C2, we have four possible combinations of condition values, like so:

C1	y	y	n	n
C2	y	n	y	n

Factoring in C3, also having two potential states, we now have eight unique possible combinations of conditions:

C1	y	y	y	y	n	n	n	n
C2	y	y	n	n	y	y	n	n
C3	y	n	y	n	y	n	y	n

We know, then, by computing the complete decision matrix, the number of columns that must be in a table to cover all possible contigencies. Referring back to our previous example, we see that although we have a total matrix of eight, as we have described, we only have four rules. To analyze this situation, we must turn our attention to a method of accounting for our "don't care" or dashed entries.

Consider rule 2:

	2
C1	y
C2	n
C3	—
A1	
A2	x
A3	

What we are really saying when we say we "don't care" about C3's value is that C3 can hold either of its values, and the rule would still hold. In other words, we can expand rule 2 into two columns.

	2a	2b
C1	y	y
C2	n	n
C3	y	n
A1		
A2	x	x
A3		

Rule 2, therefore, really contains two logical columns. It is called a complex rule. Rule 1, containing no dashes, is a simple rule. Applying the same techniques, rule 3 becomes:

	3a	3b
C1	n	n
C2	y	y
C3	y	n
A1		
A2	x	x
A3		

And rule 4, having dashes in both C1 and C2, expands to four logical columns:

	4a	4b	4c	4d
C1	y	y	n	n
C2	n	y	n	y
C3	y	y	y	y
A1				
A2				
A3	x	x	x	x

Fully expanding the whole table in this fashion, it becomes the table presented in Table 1.

After having eliminated all the dashed entries, we can count the rules and see that in our example we have nine rules, whereas our matrix arithmetic tells us we should only have eight. Examining the fully expanded table, we note that columns 4d and 3a have the same condition entries. This is also true of columns 2a and 4a.

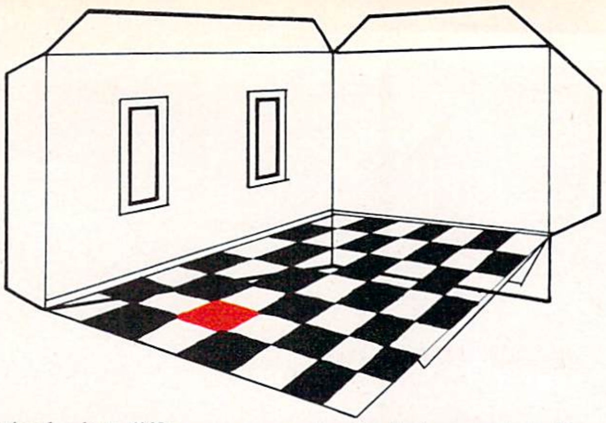
Because the interpretation of the two

	M	1	2a	2b	3a	3b	4a	4b	4c	4d
C1 Raining?	2	y	y	y	n	n	y	y	n	n
C2 Cold?	2	y	n	n	y	y	n	y	n	y
C3 Improving?	2	n	y	n	y	n	y	y	y	y
A1 Don't go out		x								
A2 Wear coat			x	x	x	x				
A3 Wait							x	x	x	x

Table 1.

	M	1	2	3	4	5
C1 Raining?	2	y	y	n	—	n
C2 Cold?	2	y	n	y	—	n
C3 Improving?	2	n	n	n	y	n
A1 Don't go out		x				
A2 Wear coat			x	x		
A3 Wait					x	
A4 Go out						x

Table 2.



sets of columns in question leads to different actions, this table is said to have contradictory rules. If 3a and 4d led to the same action(s), or 2a and 4a did, the table would be said to contain redundancies. The problem is that if you were reading the table, you could not determine which rule applies, given that their condition entries are identical.

Recall that we uncovered this problem in our table by:

- Computing the total decision matrix
- Expanding each rule that had one or more dashes
- Counting the columns in the expanded table
- Examining the expanded table for contradiction or redundancy.

If a table has as many rules as the matrix computation (M1*M2*M3 . . . Mn) says it should, and there is no contradiction or redundancy, the table is said to be mechanically perfect. That is, all possible combinations of condition values are accounted for, and none can lead to contradictory or redundant action sets.

I should point out that it is possible for the total (expanded) column count to precisely match the total decision matrix, and the table could still contain redundancies or contradictions. That's why it is necessary to expand the dashed columns and check each column against every other column, making sure there is no overlap.

Often people include a special column in a decision table, the ELSE column. If a given set of condition values do not match an explicitly stated rule, the ELSE column is invoked and whatever actions are specified underneath it are executed. A table with an ELSE rule is necessarily complete.

However, this simply begs the question. To put an ELSE rule in a decision table is like turning up the air conditioning so you're cold enough to light a fire. If you're not cold, you don't need a fire. If you want a catch-all rule, you don't really want the discipline and the benefits of a decision table structure.

For completeness' sake, let's look at our table again, perfected (Table 2). By comparing the dash-expanded table against the fully expanded matrix, we saw that we were missing rule 5 (for which action A4 was created). A systematic check of the example table against the fully expanded decision matrix to ascertain whether the example table contained all possible rules revealed that it did not.

The other bug in our table related to the dominance of rule 4, which specified that any time the weather was improving, we should wait for it to do so, and that no other condition mattered. We realized that the dashes in rules 2 and 3, C3, which could be interpreted as "y," led to contradiction. Therefore, we simplified these two rules, replacing the dashes with "n," as this was the only possibility that would both maintain the integrity of the table and give rules 2 and 3 a reason for being.

Perhaps all of this is a bit tedious. We will, in a bit, rely on our computer to relieve us of all the tedium associated with decision tables, but the point of this discussion is to introduce (for those who need it) the basic structure, terms, and use of a decision table. The important points to keep in mind are:

- The decision table is a superior communications device: the meaning is intuitively grasped.
- The attainment of mechanical perfection is possible: all possible combinations of conditions and actions can be accounted for in a systematic way.
- Latent redundancy and contradiction can be made manifest.

Applications

It should be obvious that decision tables have a high degree of usefulness in computer software documentation applications. Their communicative properties, in cases where there are complex condition/action matrices, are superior to flowcharting, pseudocode, and many of the other conventional documentation techniques.

In the case of system analysis, they are likewise valuable. The imposition of an orderly methodology to what is often a disorderly process and the real possibility of nonanalyst involvement in decision table construction combine to make decision tables especially applicable.

T.J. McCabe[2] has shown that the number of bugs in a computer program is proportional to the complexity of that program. He suggests a complexity metric, based on graph theory, that quantifies the degree of complexity in a unit of software. This metric is related to the number of branches in a software module. As a module increases in complexity (as the number of *if/then* constructs climbs), bugs proliferate.

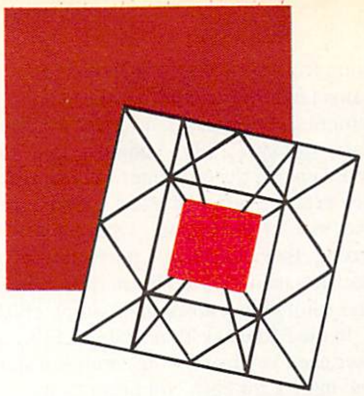
This phenomenon, which I suspect we have all experienced, is thought to be the result of the extreme difficulty most people have in keeping in mind more than six or seven things at the same time. McCabe has shown that as the number of branches in a module approaches 10, the module is increasingly unfit for use due to its predisposition for "bugginess."

This means that although it is possible for you to create a module with excessive branching, you can't really understand it because you can't fit it all into your mind simultaneously. You are therefore forced to forego the macroscopic view of your module—the relationships of all the pieces to the whole—in favor of a line-at-a-time or microscopic perspective. Naturally, if you can't understand your program, its prospects diminish.

Electronic decision tables can enable the programmer to construct vast networks of *if/then* constructs with greatly increased confidence. This is because mechanical perfection is attainable and measurable. The computer itself can ascertain whether all possible combinations of conditions are accounted for (the macroscopic view), and the programmer's inescapably microscopic view is entirely appropriate and all that is required. Thus, the computer does what it can and the programmer cannot, and vice versa. Both intelligences play to their strong suits.

This aspect of decision tables led me to construct a decision table compiler, which will be a subject addressed in Part II of this series on decision tables. Suffice it to say for now that with modest effort it can be made to compile into any high- or

THEN



low-level language or output a graphical depiction of the decision network.

If decision tables could call other decision tables, proceed from one to another, pass parameters back and forth, query operators for condition values, evaluate rules, report decision history, and do a few other lesser things, they would be the obvious medium of choice for most small to medium expert systems. If a condition stub could really be a call to a subtable, the results of which would be evaluated as the condition entry for that stub, there would be sufficient size, flexibility, and generality to cover systems of say, 500 to 1,000 complex rules. If the construction of tables were made simple and fun enough, the experts themselves would build the systems.

This notion led me to fabricate a decision table interpreter with these and other qualities. The interpreter will be addressed in the final part of this series.

Problems

A reasonable question to pose at this juncture is: if decision tables are so useful, why aren't they more universally applied? There are, in my view, two closely related reasons for this—one simple, the other a bit more complicated.

The simple one first. Spreadsheets were always used by accountant and managerial types even prior to their conversion to electronic media. They were just hard to work with. When the use of microcomputers eliminated the drudgery from and intensified the usefulness (and fun) of spreadsheets, millions of people who had never done a manual spreadsheet were drawn to the medium. My thesis is that the same phenomenon is possible, though probably on a smaller scale, with respect to decision tables.

That brings us to the second problem. What makes them so hard to use? Why haven't decision tables become a popular medium for the management of the myriad problems to which they are ideally suited?

There's nothing new about decision tables. Their history dates back to early days of computer science. Their many useful properties have been thoroughly investigated, their applicability to computer-related operations has been identified, decision table symposia have even been held, decision table books have been published, and yet decision tables remain something of an obscurity, their use seemingly confined to a relatively small group of devotees.[3]

Anyone who has spent much time in working with decision tables realizes that while they are a wonderful way to communicate information, both to humans and computers, they are a difficult way to express it. The mental steps it takes to create, edit, check, and perfect a decision table are painful to most minds, alien to others, and can be ultimately forbidding in all but the most trivial of examples.

Rather than the simpleminded example above, picture six conditions, each with a matrix of five values. That translates to over 3,000 distinct possible simple rules. And yet many real-life cases have at least that much complexity.

The difficulty of resolving a problem of this magnitude into a workable number of interrelated conditions and actions, the tedium of developing rules for all possible combinations of conditions, and the complexity of checking for completeness, redundancy, and contradiction apparently overwhelm the perceived value of doing so. Therefore, decision tables— which really are a most powerful way of encoding, communicating, and using many types of intelligence—languish.

This, of course, was the main problem that confronted me when I thought of all the benefits that would accrue to me if only I would consistently build decision tables. The fact is, they are hard to make.

Table editor

So I decided to make it easier to work with decision tables. The goals were various, but of one thing I was sure: the problem was in the creation and perfecting of tables and not in their application. My principal design focus was therefore on the creation of an interactive decision table editor with a high degree of tedium-relief.

The first tedious task that was to be eliminated was the fabrication and keying in of condition entries. It is easier to respond to or change something that already exists than it is to create some-

EXSYS

A LOW COST EXPERT SYSTEM DEVELOPMENT TOOL

Create knowledge based, probabilistic IF-THEN rule expert systems on an IBM PC, XT or compatible

Provides the user with a list of possible solutions to a problem arranged in order of probability

Performs backward chaining and can explain how conclusions were reached and why information is needed

External programs, including most spread sheets and data base managers, can be called with data passed into EXSYS for analysis

ALL input is English text, algebraic expression or menu selection

Numeric variables are supported and can be calculated and displayed by the expert system

Full available memory is used and the programs are written in C for fast, efficient code

700 rules per 64k of memory over 192k (almost 5000 rules in a 640k PC)

Trig, log, exp and sqrt functions supported

Developed expert systems marketable **WITHOUT ROYALTY**

Requires IBM PC, XT, AT or compatible with DOS 2.0 or higher and 256k

DEMO DISK — Demo knowledge bases, text of manual, automated instruction on how to use the program and ability to create and run rules – $10 (refundable with purchase)

FULL PROGRAM — $295 + $5 postage and handling

MACINTOSH VERSION AVAILABLE SOON

EXSYS INC, P.O. BOX 75158
CONTR. STA. 14, ALBUQUERQUE, NM 87194
(505) 836-6676

CIRCLE 54 ON READER SERVICE CARD

with

TURBO REF

a Cross-Reference and Lister utility for Pascal source programs.

- All options user-selectable.
- Identifies line number for each variable reference.
- Can cross-reference constants.
- Indicates type of reference.
- Will process a list of files.
- Will read Include files.
- Draws boxes for control structures — example:

```
if PascalProgrammer then
begin
  CallGracon;
  repeat
    RunTurboRef;
  until Organized;
end;
```

Written in Turbo Pascal for the IBM PC. PC-DOS 2.0 and 128K required.

Only $49⁹⁵

MC, Visa, or company check.
C.O.D. orders add $4.00.
Michigan residents add $2.00 sales tax.

GRACON SERVICES, INC.

VISA

4632 Okemos Rd.
Okemos, MI 48864
(517) 349-4900

MasterCard

IBM and PC-DOS, and Turbo Pascal are registered trademarks of International Business Machines Corp. and Borland International Inc., respectively.

CIRCLE 56 ON READER SERVICE CARD

thing from nothing. This element of human nature is a guiding principal in the integration of user friendliness in computer software.[4] My vision was that if I could enable the computer to perform the raw creative task and make it easy for the user to tell the computer where it was wrong, the creation and perfection of decision tables would be an infinitely simpler (though not absolutely simple) task.

In the ERGO system's table editor, the user need key in only the condition stubs and matrix for each condition (if it is greater than two) and the table editor will generate a mechanically perfect set of simple (no dashes) rules. The user then need only edit this data and enter the appropriate action stubs and entries.

The table editor will also kill redundant or contradictory rules, so that the creation of a table is really a process of editing a computer-generated rule, usually by adding dashes and then letting the table editor eliminate the resulting superfluous column(s).

This describes the bare rudiments of the ERGO system's table editor operation. Many more features and facilities will be described later, along with some of the associated code.

Grid chart editor

Sometimes it is convenient to create logical structures in a different way, namely, with the action as the key element:

Wait for weather improvement when
  it's forecasted to improve.
Wear your coat when its raining or
  cold.
Don't go out when the forecast is bad.
Go out when its not raining or not cold
  or not improving.

Notice how much simpler this expression is than the lengthy description of our earlier example. This is because when you express logic keyed by action, you need never have more clauses than actions. This simple technique can be an appealing way of describing condition/action relations because of its inherently limited boundaries. Where our condition-keyed example had multiple clauses and was still incomplete and redundant, the action-keyed description of the same problem is terse and complete.

The pertinent point here is that a problem so expressed can be transcribed directly into an entity called a decision grid chart, which can itself be converted into a mechanically perfect decision table. This can give a user a tremendous leg up on the creation of a logically perfect decision table. Extending our commitment to the principle that it is easier to edit than to create, I wrote a grid chart editor that works in tandem with the other pieces of the kit.

By simply expressing the condition values that imply the actions, paying no attention to the combinations of conditions, a mechanically perfect and logically close decision table, with both condition and action entries, can be automatically produced. From there we can edit, compile, interpret, or depict.

An interesting aspect of decision grid charts is their close relation to common English. Though I haven't done the programming, the translation of English language statements in a pre-grid chart format to grid charts seems clearly possible.

Part II and III

In Part II of this series, we will be examining the mixed entry table editor, for this is the most fundamental piece of the decision table system—the ERGO Logic Kit—to which I have been alluding. We also will examine the compiler, algorithms and code for automatic condition entry, finding missing rules, eliminating redundancy and contradiction, automatic table compression, sorting and optimizing tables, and other interesting topics.

In Part III we will explore the decision table interpreter as well as look into the grid chart editor and translator and other related topics such as a natural language front-end and factoring probabilities into decision tables.

See you next month! ■

References

1. Kay, Alan. "Computer Software." *Scientific American* (Sept. 1984).
2. McCabe, T.J. "A Complexity Measure." *IEEE Transactions on Software Engineering SE-2* (1976): 308-320.
3. A great deal of literature is available on the subject of decision tables. I would suggest the following materials for an introduction: Hurley, Richard B. *Decision Tables in Software Engineering*. New York: D. Van Nostrand Co. Inc., 1983. Pooch, Udo W. "Translation of Decision Tables." Vol. 3, No. 6, *Computing Surveys* (June 1974).
4. Smith, David Canfield, Charles Irby, Ralph Krimball, Bill Verplank, and Eric Harslem. "Designing the Star User Interface." *BYTE* (Apr. 1982).

Jim McCarthy is a member of the technical staff, research and development, AT&T-Teletype, in Skokie, Ill.

Introducing the MIX Editor

(with Split Screen - both horizontal and vertical)

A Powerful Addition To Any Programmer's Tool Box

Full Screen Editing WordStar Key Layout Custom Key Layouts Terminal Configuration Help Files Backup Files	*Introductory Offer Only* **29.95** 30 Day Money Back Guarantee	Programmable Macro Commands Custom Setup Files Mnemonic Command Mode Multiple File Editing Split Screen Editing

For PCDOS/MSDOS (2.0 and above/128K) • IBM PC/Compatibles, PC Jr., Tandy 1000/1200/2000, & others
For CPM80 2.2/3.0 (Z80 required/64K) • 8″ SSSD, Kaypro 2/4, Osborne I SD/DD, Apple II, & others

Great For All Languages

A general purpose text processor, the MIX Editor is packed with features that make it useful with any language. It has auto indent for structured languages like Pascal or C. It has automatic line numbering for BASIC (255 character lines). It even has fill and justify for English.

Terminal Configuration

A utility for defining terminal features (smart features included) allows the editor to work with any terminal. Over 30 of the most popular terminals are built-in.

Custom Key Layouts

Commands are mapped to keys just like WordStar. If you don't like the WordStar layout, simply change it. Any key can be mapped to any command. You can also define a key to generate a string of characters, great for entering keywords.

Split Screen

You can split the screen horizontally or vertically and edit two files simultaneously.

Macro Commands

The MIX Editor allows a sequence of commands to be executed with a single keystroke. You can define a complete editing operation and perform it at the touch of a key.

MIX software 2116 E. Arapaho Suite 363 Richardson, Tx 75081 (214) 783-6001

MSDOS is a trademark of Microsoft
PCDOS is a trademark of IBM
CPM80 is a trademark of Digital Research
WordStar is a trademark of MicroPro

Custom Setup Files

Custom keyboard layouts and macro commands can be saved in setup files. You can create a different setup file for each language you use. The editor automatically configures itself using a setup file.

Command Mode

Command mode allows any editor command to be executed by name. It is much easier to remember a command name versus a complicated key sequence. Command mode makes it easy to master the full capability of the editor. Frequently used commands can be mapped to keys. Infrequent commands can be executed by name.

Editor Commands

The editor contains more than 100 commands. With so many commands, you might think it would be difficult to use. Not so, it is actually extremely simple to use. With command mode, the power is there if you need it, but it doesn't get in your way if you don't. Following is a list of some of the commands.

Cursor Commands

Left/Right/Up/Down
Tab Right/Tab Left
Forward Word/Backward Word
Beginning of Line/End of Line
Scroll Up/Scroll Down
Window Up/Window Down
Scroll Left/Scroll Right
Top of File/Bottom of File
• • •

Block Commands

Copy/Move/Delete
Read/Write
Lower Case/Upper Case
Fill/Justify
Print

File Commands

Directory (with wild cards)
Show File/Help File
Input/Output File
Delete File/Save File

Other Commands

Split Screen/Other Window
Find String/Replace String
Replace Global/Query Replace
Delete Line/Undelete Line
Delete Word/Undelete Word
Insert Mode/Overwrite Mode
Open Line/Join Line
Duplicate Line/Center Line
Set Tab/Clear Tab
• • •

To Order: **Call Toll Free 1-800-523-9520,** (Texas only **1-800-622-4070)**
Mix Editor ____ $29.95 + shipping ($5 USA/$10 Foreign) Texas residents add 6% sales tax

Visa ____ MasterCard ____ Card # ________________ Exp. Date ____________
COD ____ Check ____ Money Order ____ Disk Format ________________
Computer ________________ Operating System: MSDOS ____ PCDOS ____ CPM80 ____
Name ________________
Street ________________
City/State/Zip ________________
Country ________________
Phone ________________

MIX software 2116 E. Arapaho Suite 363 Richardson, Tx 75081
Dealer Inquiries Welcome
Call (214) 783-6001

BT

CIRCLE 24 ON READER SERVICE CARD

We're getting hardnosed at Softway.

From now on MATIS™ is only 49^{95}!

(MATIS, the complete User Interface development tool has been selling for $150.)

Why the radical price cut?
We decided after looking over the competition that MATIS had so many advantages it should be made available to more programmers. We decided to compete aggressively so you could easily afford to have MATIS in **your** bag of tricks. We hear from MATIS users in the USA and France that it is a truly loveable product. Sooo...we're running this big ad to promote our new low price.

MATIS windows are beautiful.
Display any portion of any screens you create at any point in your program. Scroll in any direction manually with cursor keys...or automatically.

And the screens are HUMUNGEOUS!
MATIS screens can be just about as big as you want...up to 65,534 rows by 65,534 columns! The number of screens is only limited by available memory.

Print big MATIS screens directly.
One command sends your screens to your printer with no need to program special routines when your virtual screen is bigger than your terminal screen.

User input fields are a snap.
Creating fields for data entry is easy with no limit to size or number by screen. Request for input separately or in blocks.

Denis Moran
President, Softway, Inc.

Control your keyboard with MATIS.
It keeps track of keys that are pressed during the execution of your program and lets you assign specific functions to selected keys.

Control that screen too!
MATIS is extremely versatile and flexible when it comes to controlling lines, columns, fields, and text. They can be modified, transferred, displayed or moved with a single command. All video attributes are supported: color, reverse video, blinking...you name it, you got it.

Want an interactive screen builder?
You've got it with MATIS. It's called "MATPAGE"™ and it lets you create and modify any of your screens in an interactive mode.

MATIS adds over 70 routines to your program.
Written in Assembler, MATIS routines are fast and powerful giving your program improved efficiency and enhanced visual appeal, while they reduce its size and maintenance worries. And MATIS separates screen design from the core of your program.

MATIS is unique.
We don't think there's a single program that combines as many tools in one package as completely or as well as MATIS. It interfaces with Intepreted and Compiled BASIC (Microsoft), C (Lattice, Microsoft, Aztec), PASCAL (IBM, Microsoft) and ASSEMBLER. All you need is an IBM* PC/XT or true compatible under DOS, 128k or RAM, monochrome or color monitor.

You get an easy to follow no-frills manual and a 30-Day Money Back Guarantee.

Late News:
MATIS/T™ for TURBO-PASCAL only $29.95**
An indispensable add-on at a dynamite price. What more can we say?

Denis Moran

Denis Moran

™MATIS, MATIS/T, & MATPAGE are Trademarks of Softway, Inc.
*IBM is a Reg. Trademark of IBM Corp.
**Turbo Pascal is a Reg. Trademark of Borland International

Softway, Inc.

500 Sutter Street, Suite 222 CG
San Francisco, California 94102
(415) 397-4666

24-Hour Credit Card Orders By Phone:
1 (800) 227-2400 EXT 989 In California: 1(800) 772-2666 EXT 989

Please ship the following at once. I understand there is a 30-day money back guarantee.

_______ Copies of MATIS at $49.95 plus $3 shpg. $ _______
_______ Copies of MATIS/T for TURBO PASCAL at $29.95 plus $2 shpg. $ _______
California residents, add 6½% sales tax $ _______
☐ I like to read specs, so send me a folder. Total $ _______

Name _______
Address (please no P.O. Box) _______
City/State/ZIP _______
Phone (   ) _______ Signature _______
Make payment by money order, check, or charge card ☐ VISA ☐ MASTER CARD
Number _______ Exp. _______

Distributed in Europe By: MICRO APPLICATION SOFTWARE • 147 Avenue Paul Doumer, 92500 RUEIL MALMAISON FRANCE, Tel (1) 732.92.54

CIRCLE 29 ON READER SERVICE CARD

Programming techniques and products that put English on your PC

Natural Language Software

By Darryl Rubin

You probably remember the movie "2001" for its artificially intelligent computer, HAL. Not only was HAL a smooth talker, he was a canny listener, able to understand and obey spoken commands.

With the real 2001 just 16 years away, are we much closer to building computers that can communicate as naturally as HAL? Might the sixth generation computer language be English? Let's consider.

Computer scientists have made great strides in natural language processing (NLP) since "2001" was released. Programs have been demonstrated that can translate natural languages,[1] paraphrase simple texts,[2] even read and answer questions about AP news stories.[3] Natural language data base systems are especially popular items, and not just in the lab. Try your nearest mainframe computer, where natural language query products like Intellect[4] have been available for years.

Or try your personal computer. You can buy natural language data bases for your PC or if you're ambitious, implement one yourself.

In this article we'll look at the problems NLP poses and discuss some techniques for solving them. We'll explain a popular method for parsing English sentences. We'll also take a hands-on look at two natural language data bases for PCs that show just how far NLP has come and suggest how far it may yet go. The problem of implementing HAL we'll leave as an exercise for the reader.

Natural history

NLP research is older than you might think. People were experimenting with natural language software before HAL was even a gleam in Stanley Kubrick's camera lens.

The earliest efforts[5] focused on the problem of determining the grammatical structure of English sentences, a process you compiler writers will know as parsing. These early programs did nothing more than diagram input sentences, producing parse trees like the example in Figure 1.

Recent programs are much more impressive because they attempt to understand and act on the actual *meaning* of the input sentences. One program, SHRDLU,[6] could move objects about a simulated world based on English requests; the program could also understand and correctly answer questions about the current state of its world. Another program, SAM,[7] could read simple stories and answer questions about them, in many cases inferring information that wasn't explicitly stated.

A parse tree

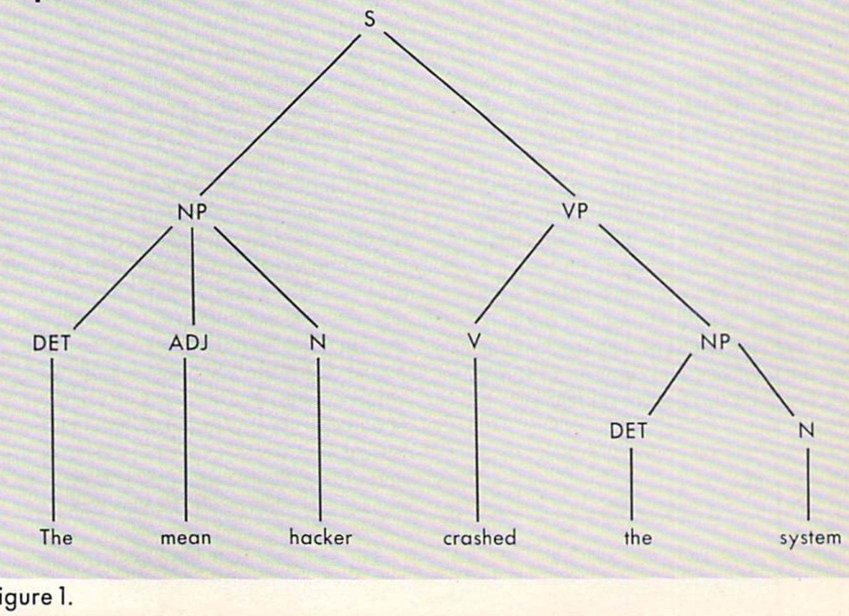
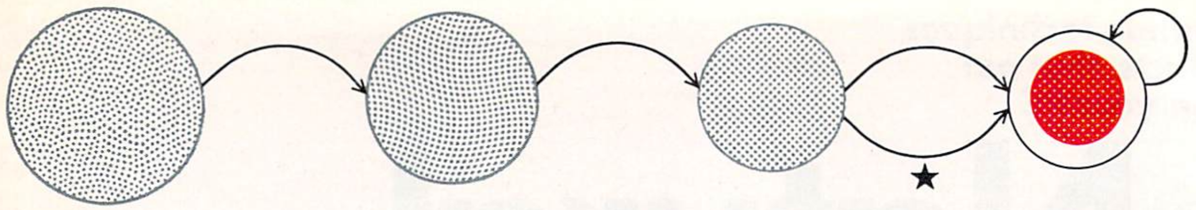


Figure 1.



Understanding natural language is no small feat. Language is rich in ambiguity, innuendo, and sometimes just plain sloppy grammar. Indeed, human beings often have trouble understanding it! The first step then, is to simplify the NLP problem by breaking it down into several smaller problems, or phases.

If you've ever written a compiler, you'll know right off that NLP can be divided into three phases: lexical, syntactic, and semantic analysis. You start with an input string of one or more English sentences. Lexical analysis separates the string into individual words and classifies them according to a dictionary. Syntactic analysis groups the words into phrases and phrases into sentences. Semantic analysis figures out what the sentences mean.

Some linguists will point out that true NLP requires a fourth phase called pragmatics that deals with the overall communicative intent of a dialog. For example, the statement "That's wonderful" means one thing on the surface but quite another if it is a response to "Taxes just went up." The difference is one of pragmatics.

As you can imagine, semantic and pragmatic analysis of natural language are unsolved problems. Let's just get a feel for them before moving on to the more tractable lexical and syntactic phases. Look at these two sentences:

Babe Ruth threw a ball.
Queen Victoria threw a ball.

Syntactically these sentences are identical and unambiguous. But how different the meanings of the words "threw" and "ball."

Ambiguity can, of course, be a problem, as in "I saw Nero on the stage with binoculars." Think carefully: Who has the binoculars, me or Nero? For that matter, which of us is on the stage? Also ask yourself, would a computer understand that Nero must be a character in a play, not the historical emperor of Rome? Now consider this pair of sentences:

Nothing is better than complete happiness in life.
A ham sandwich is better than nothing.

Unless a computer is awfully smart about interpreting these sentences, just image the conclusion it's likely to draw!

Lex education

Perhaps a HAL could deal with problems like these, but practically speaking, you can implement useful NLP programs just by doing a good job at lexical and syntactic analysis. This is because most programs are written to solve specific problems, like data base inquiry, and in these limited domains the range of possible meanings is small. For example, data base query systems deal only with variations on a single meaning: find information in the data base.

Whereas HAL could understand spoken inputs, most of today's NLP programs can only deal with written input—text strings. We said earlier that lexical analysis is the first step in processing natural language. The lexical analyzer breaks the input string into its component words and important punctuation marks, like sentence endings. It then looks up the words in a dictionary to check whether they are known and if so to attach important features to them, like parts of speech—article, noun, adjective, verb, etc. (or several of these for words with multiple meanings). Words not found in the dictionary may be removed from the input or spell correction may be attempted.

Identifying word and sentence endings may sound simple enough until you realize how creatively the rules of punctuation can be applied by some writers. Also consider the problem that abbreviations, contractions, and compound words can create. Abbreviations end in a period, which may look like a sentence ending. Contractions use apostrophes, which are also used in quotations and possessives. Compound words use a hyphen, which can be confused with a word break at a line boundary. Try this sentence out on your lexical analyzer: "Don't tell me the computer 'eats' double-sided diskettes—that's nonsense!" said the Jr. Programmer's boss.

The easiest way to handle difficulties like these is with a good dictionary. For example, the dictionary can be used to expand abbreviations and contractions to full words or to disambiguate the use of hyphens by checking whether removing the hyphen results in a valid word. The dictionary is less useful for disambiguating the different uses of apostrophes, so heuristics must be developed to do this job.

Another tactic that works for some applications is to simply ignore punctuation. Natural language query systems typically do this, including the ones we'll talk about later. But a general-purpose understander can't take it quite that easy. Try making sense of "That's what I meant you thought." Confusing, yes? But add just one little comma and a plausible meaning emerges: "That's what I meant, you thought."

The dictionary can be used to simplify input text in more profound ways than merely eliminating abbreviations and contractions. For example, certain entries can be marked as "noise words" that are to be removed from the input text; other words or entire phrases can be marked for replacement by predefined synonyms. Combined with an algorithm for removing suffixes and prefixes, the dictionary can also help transform words to their root forms. Simplifications like these remove unnecessary detail and make the input text easier to analyze.

In some applications you can even shrink sentences to a few key words that convey all the needed information. Many adventure games work this way, isolating a single verb and object out of the player's input. "Let's drop the shovel here" becomes "Drop shovel."

Another program that uses such techniques is a full text retrieval system called SIRE. Similar in concept to the Dialog Information Service, SIRE lets you store documents on disk and retrieve them based on queries like "Show me articles about natural language" or "Have you got anything about cognition dated 1985?" SIRE doesn't actually parse or understand natural language. But by clever noise-word filtering, root-word extraction, and synonym lookup, it reduces queries like the two preceding ones to the essential retrieval keys "natural language" and "thinking or cognition, date 1985."

Grammar school

The syntactic phase of NLP is the one you probably learned the most about in school. It's what 12 years of English class were all about.

Syntactic analysis concerns itself with the rules of grammar—that is, the rules for combining the basic parts of speech into phrases, clauses, and sentences. The problem for NLP is to use these rules to analyze an input sentence and produce a parse tree representing the sentence's structure. Figure 1 shows a parse tree for the sentence "The mean hacker crashed the system."

The hardest part of syntactic analysis is to encode the grammatical rules that guide the building of parse trees. Natural languages have very complex grammars that are context dependent, not context free like those for computer languages. This means that the role one word plays in a sentence depends on other words in complex ways. Take this example:

Time flies like an arrow.
Fruit flies like a banana.

In the first sentence, "flies" is a verb and "like" is a preposition. In the second, "flies" is a noun and "like" is a verb. How can a parser analyze sentences like these? First, it must depend on the lexical analyzer to mark each word with every role the word can play. Second, the parser may need to backtrack—that is, when finding an ambiguous word, pick the most likely meaning and proceed with the parse. If the parse fails, back up and retry with another of the possible meanings.

The more you think about complexities like this, the more you realize that a natural language parser is a kind of expert system whose knowledge base is a set of grammar rules. How can these rules be encoded? The most well-proven method is to use augmented transition networks (ATNs),[8] like the ones shown in Figure 2.

ATNs will look familiar if you've ever seen the syntactic flow diagrams used to specify computer languages like Pascal. But there's more to ATNs than meets the eye. Let's see how they work.

The ATN in Figure 2A says that a sentence (S) consists of a noun phrase (NP) followed by a verb phrase (VP). This is because crossing the NP arc followed by the VP arc leads to a success node (indicated by double circles).

But what is a NP? It is another ATN (Figure 2B), which says that a NP consists of an optional determiner (DET) followed by zero or more adjectives followed by one or more nouns—for example, "The (DET) lazy (ADJ) computer (N) operator (N)."

Using an ATN is like finding a path through a maze. If you can get to a success node, the parse succeeds (the sentence was grammatical), otherwise it fails (the sentence was probably everyday English!).

So far, ATNs sound just like syntactic flow diagrams. What puts the A in ATN is that each arc is augmented with a procedure that must be executed to determine whether the arc is currently available for crossing. These arc procedures can inspect the input string, modify it, even maintain local and global state informa-

ATNs for a subset of English grammar

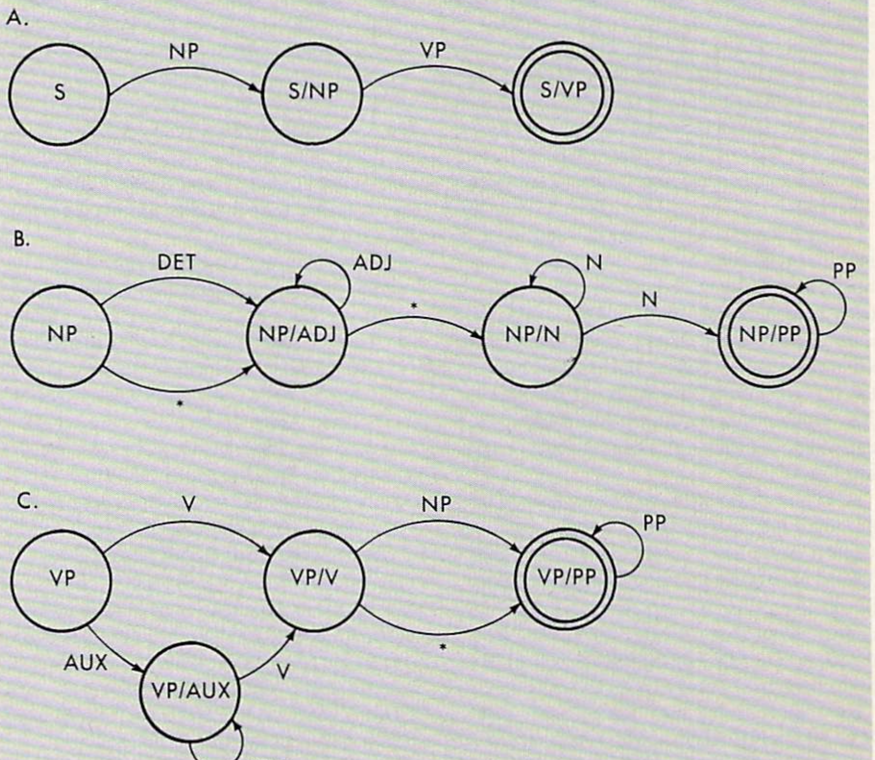
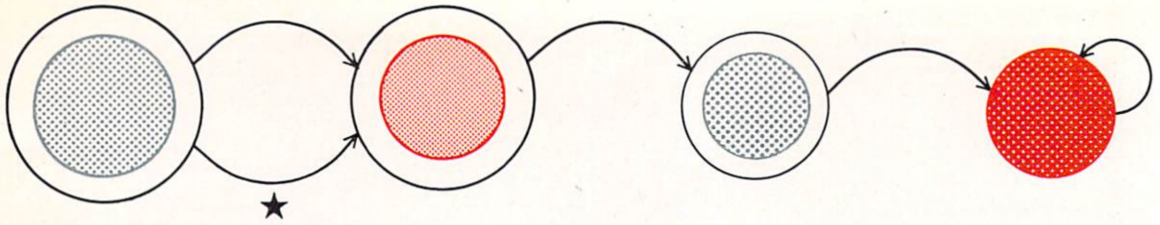


Figure 2.



tion in registers that are totally separate from the transition network itself.

Most importantly, the arc procedures are responsible for building the parse tree a node at a time as the network is traversed. They use the registers to do this. Registers are usually defined that correspond to major sentence features like the subject, object, verb, tense, and so on. As the parser classifies words in the input string it appends them to the appropriate registers, then assembles the register contents into parse tree nodes when the necessary preconditions for doing so have been satisfied (such as reaching a success node of the ATN).

To move from one node to another, the parser must evaluate (call the procedure of) each outgoing arc to determine which one is available for crossing. For example, the arcs labelled N in Figure 2B can only be crossed when the current input word is a noun. Likewise, the ADJ arc requires an adjective, PP arcs require a prepositional phrase, V requires a verb, and so on. (Arcs labelled "*" are always available for crossing—they provide a way to make certain grammatical features optional.)

Generalizing on this notion, an arc labelled with the name of another ATN requires that the specified ATN be traversed successfully. In other words, the named ATN is called as a (possibly recursive) procedure. Notice in Figure 2 how the NP ATN calls PP which in turn calls NP. This mutual recursion could conceivably occur for call after call after call after call!

Of course, not all kinds of repetition are legal in English. For instance, double negatives are a no no (ahem). How can the VP ATN (Figure 2C) detect them? Simply by setting a flag register in the *NEG* procedure to bar that arc after the first time through. The VP ATN must also check for invalid verbs forms; it's OK to write "have goofed" but not "have goof." The V arc from NP/AUX to VP/V tests for this mistake—it blocks itself if "have" was seen as an auxiliary verb (AUX), but the current word is not a past participle.

When checking for available arcs, the parser may find itself with several that can be taken. The NP/N node in Figure 2B presents one such dilemma: it has two N arcs leaving it. When this happens, the parser must backtrack as previously explained, trying each of the alternatives until one succeeds or all fail. This gets slow in a hurry, so be sparing of multiple choice nodes in your own ATNs.

A Savvy Retriever

With all the effort that's gone into refining ATN parsers and other NLP algorithms, it should come as no surprise that the technology has begun appearing in microcomputer products. One of the first of these is a natural language query system called Savvy Retriever.

The surprising thing about Savvy is that it uses none of the established NLP techniques. Savvy's approach is Savvy's own. And there's a good reason for this.

Classical parsers are built to process grammatically correct sentences and to reject incorrect ones. But how many of us are grammatically correct all the time? We use flaky syntax, fragmentary sentences, misspelled words, bad punctuation, you name it. The most convoluted command language would probably be easier to use than one that required perfect English!

Savvy's unique approach is to dispense with parsing almost entirely. Instead, this program relies on a technique called adaptive pattern recognition. Rather than analyze your query's grammatical structure (which may not exist), Savvy asks itself "Of all the queries I've ever seen, which one does this most resemble? Is there a pattern here that I recognize?" If so, Savvy assumes the new query is equivalent to the one it had seen before. The program will optionally display the formal interpretation of your query so you can verify it. To get an idea of how Savvy works in practice, a complete, annotated transcript of a Savvy work session is available as SAVVY.LOG on the *COMPUTER LANGUAGE* Bulletin Board Service or account on CompuServe.

Savvy's pattern recognition process is fascinating. The program operates on your query's representation as a string of bits in computer memory—not sentences, not words, *bits*. Through a series of mathematical transformations that are analogous to hash coding, Savvy derives from this bit string a set of numbers that represents a coordinate in an abstract space of 200 dimensions (!).

Savvy's pattern memory consists of clusters in this space, each of which represents a specific word, phrase, or query that it understands. Pattern matching is simply a matter of seeing if your query's spatial coordinate lies within any of the prestored clusters. (These clusters are created at the factory in a training process where Savvy is presented with sets of different queries that mean the same thing.) If Savvy can't match your query as a whole, it splits the query into smaller parts and pattern matches those, recursively to the word level if necessary.

The beauty of this approach is that Savvy is incredibly tolerant of misspellings and excess verbiage. More often than not, it focuses on the significant content of your query and within limits (you'll see later) produces the desired information.

Like its strengths, Savvy's drawbacks result from its pattern matching algorithm. Pattern matching is not the same thing as understanding. Savvy cannot perform calculations because it cannot parse expressions; you can't ask it to average a data base field, for example. This lack of expression handling also makes Savvy weak at handling complex Boolean retrieval conditions, as in "List all employees who are under thirty five and single or over forty and married." Worse, the pattern matcher may misinterpret queries. For example, it may interpret "List recently hired employees" as "List recently fired employees."

Another weakness is that Savvy's pattern memory can only be trained at the factory (the process involves massive number crunching). To make up for this, Savvy lets you define literal synonyms for words it already understands; you may define synonyms for any of Savvy's 85 built-in words or for any field label or data value in your data base. Unfortunately, you cannot define synonyms for phrases. For example, you can't define "highly paid" to mean "salary greater than 40000"; Savvy will insist you pick between defining a synonym for "salary" (a field label) or "greater than" (a built-in operator).

Queries with Clout

Savvy's holistic, pattern matching approach to NLP contrasts sharply with the analytical, logic-based technique used by Clout, another natural language query system for PCs. Much more than a parser, Clout is an expert system that understands how people ask questions. And it understands very well indeed. A complete, annotated Clout session is also available on the *COMPUTER LANGUAGE* BBS and on CompuServe—look for the file name CLOUT.LOG.

Like Savvy, Clout is tolerant of ungrammatical, fragmentary, misspelled queries. But there the similarity ends. Clout analyzes your query to the *N*th degree, testing it against a knowledge base of *if/then* rules that encode expertise about data base queries. For example, one of Clout's rules says "If the word TOP is present, the phrase following it is probably a ranking criterion." So it understands "What are the top 3 salaries?"

Via such rules, Clout tries to assign meaning to every word in your query. If it runs into trouble, the program uses ATNs to provide additional information and may even ask you to define unknown words. Clout's parser understands mathematical expressions as well as English grammar, so Clout can perform calculations ("What is Jones' salary * 1.10?").

Unlike Savvy, when Clout doesn't understand something it immediately gives you a list of guesses to choose from or asks you to enter a synonym. This makes it a snap to adapt Clout to your question asking style. Clout is much more trainable than Savvy because it has a larger vocabulary (300 words) and a more powerful synonym capability. What's more, you can define synonyms for entire phrases, and the definitions can take parameters. This lets you define "works for @X" as "supervisor = @X" so you can ask "Who works for Bronson?"

Clout draws knowledge not just from its dictionary and knowledge base but also from your data base. Clout knows how to use the data base schema to retrieve information from up to five files. The program automatically performs whatever data base operations are needed (select, project, join) to retrieve the desired information.

Unfortunately, Clout can't show you the formal representation of your query as Savvy can. This is because Clout translates your query directly into data base I/O operations. In fact, the data base I/O interface that Clout uses is available separately from Microrim as a product called PI (Program Interface). Clout itself is just a big Fortran program (!) written on top of the PI.

Natural selection

Clout and Savvy represent impressive first steps toward NLP on personal computers. They also demonstrate that the first two decades of natural language research didn't exhaust all the good ideas. Savvy's pattern recognition algorithm works well for simple but imprecise queries against single files. Clout's expert system approach makes short work of complex, multifile queries and calculations.

These two products and their successors will surely evolve, not just in terms of their ability to understand, but also in terms of their ability to act. Will future data bases perform natural language updates as well as queries? Will future word processors use NLP to rephrase poorly written passages and summarize documents for quick reading? Will your future computer answer questions as smoothly and intelligently as a HAL? Perhaps.

If it doesn't start asking *you* questions first. ■

References

1. Wilks, Y. "The Stanford Machine Intelligence Project." In R. Rustin (Ed.), *Natural Language Processing*. Englewood Cliffs, N.J.: Prentice-Hall, 1973.
2. Schank, R., N. Goldman, C. Rieger, and C. Riesbeck. "Inference and Paraphrase by Computer." *JACM* 22(3) (1975): 309-28.
3. Schank R., M. Lebowitz, and L. Birnbaum. "An Integrated Understander." *American Journal of Computational Linguistics* 6(1) (1980).
4. Harris L. "A High-Performance Natural Language Processor for Data Base Query." *ACM Sigart Newsletter* vol. 61 (1977).
5. Feigenbaum E. and J. Feldman (Eds.). *Computers and Thought*. New York: McGraw-Hill, 1964.
6. Winograd, T. *Understanding Natural Language*. New York: Academic Press, 1972.
7. Lehnert, W. "Human and Computational Question Answering." *Cognitive Science* 1(1) (1977): 47-73.
8. Johnson, R. "Parsing with Transition Networks." In M. King (Ed.), *Parsing Natural Language*. New York: Academic Press, 1983.

Darryl Rubin is section manager for network products at ROLM Corp.

Natural language products

Product: SIRE
Price: $600, demo disk $25 (IBM PC version)
Availability: IBM PC, XT, AT, or compatibles, DEC Rainbow, computers running UNIX or RSX
Vendor: Cucumber Information Systems, 5611 Kraft Dr., Rockville, Md. 20852, (301) 984-3539

Product: Savvy/PC with Savvy Retriever
Price: $395
Availability: IBM PC, XT, or AT running PC-DOS 2.0 or higher
Vendor: The Savvy Corp., 800 Rio Grande Blvd., N.W., Albuquerque, N.M. 87104, (800) 551-5199

Product: Clout
Price: $250. Options: Rbase 4000 ($495), Program Interface ($395)
Availability: IBM PC, XT, AT, or compatibles running PC-DOS or MS-DOS 1.1 or higher
Vendor: Microrim Inc., 3380 146th Place, S.E., Bellevue, Wash. 98007, (206) 641-6619

QUINTUS

Computer Systems Incorporated

TM

The Logical Solution for Advanced Software Development

Quintus Prolog™ Release 1.0

Available on... VAX (UNIX 4.2 & VMS), Sun Microsystems (UNIX 4.2) ...and more to come
Fastest commercial Prolog
Most efficient memory utilization
Incremental Compiler
Integrated full screen editor
Comprehensive debugger and style checker
On-line help and documentation
Interface to C, Fortran, Pascal & VMS supported languages
Floating point arithmetic
Complete documentation, technical support and training available

Quintus™ FIRSTS!!!

First Prolog to exceed 20,000 LIPS on a workstation

First available Prolog developed specifically for commercial use

First Prolog to achieve 1 LIP per dollar cost performance

First symbolic language with compatible compiler/interpreter

The Company Foundation

Dr. David Warren Authored first Prolog compiler: DEC 10. The "Warren" in Warren/Tick Machine.

Dr. William Kornfeld Expert in Prolog *and* Lisp technology. Designed Quintus Prolog development environment.

Lawrence Byrd Developed DEC 10 debugger and Quintus Prolog language interface.

Dr. Fernando Pereira Authored C-Prolog. Natural language expert.

Tom Hartnett President and CEO. 17 years of industry and management experience including compiler and programming tools for micros through mainframes.

Dr. Cuthbert Hurd Instrumental in IBM's decisions to develop their first computer, Fortran and transistor-based computers.

Sample Quintus Prolog Applications

CAD/CAM interface (Natural Language)
CAD/CAM research
Chip design verifier
Configurator/expert system
Database and software technology research
Expert system prototype development
Expert systems and natural language
Machine learning and algebraic manipulation
Natural language front end to database
Teaching and research

Technology Established *Product Delivered* *Satisfied Customers* *Sound Financial Status*

Quintus Is Actively Hiring: Software Engineering, Quality Assurance, Technical Documentation, Sales, Technical Support, Finance and Accounting

Please Contact:
Quintus Computer Systems, Inc.
2345 Yale Street, Palo Alto, CA 94306
Telephone (415) 494-3612
Telex 9103805744 QUINTUS

European Distribution:
Artificial Intelligence Limited
Intelligence House, 62-78 Merton Rd.,
Watford, Hertfordshire, WD1 7BY
Telephone (0923) 47707
Telex 851933883 INTHSE G

Learning about PROLOG

By Ramachandran Bharath and Margaret Sklar

In 1981 it was announced that Japan had chosen the programming language PROLOG as the basis for software to be developed in the Fifth Generation Computer Project. One of the aims of this project was to make software that would be truly handy for users.

The choice of PROLOG caused a great deal of attention to be focused on the language. Until then, it had been a research language, first developed at Marseilles, France, and later extended at Edinburgh, Scotland, and other centers.

The purpose of this article is to give an indication, through simple examples in PROLOG, of what is distinctive about the PROLOG programming style and why it is especially suitable for user-friendly applications.

A number of versions of PROLOG are available, with some differences in syntax. But generally, the accepted standard seems to be the version given in the first definitive textbook on PROLOG, *Programming in PROLOG* by W.F. Clocksin and C.S. Mellish. This is essentially the Edinburgh University version of PROLOG.

The examples in this article will be mostly in terms of the C & M standard version, which, with some minor modifications, is used by most mainframe and micro versions of PROLOG. We will also illustrate the Micro-PROLOG version of PROLOG, which is put out by Logic Programming Associates for IBM PCs and Apple computers using CP/M.

A deductive data base

PROLOG is a portmanteau word derived from PROgramming in LOGic. A PROLOG program consists of declarations of relationships between objects. During program execution, PROLOG draws logical deductions from facts or relationships the user has supplied. This process makes PROLOG distinctly different from other programming languages.

The key characteristic of other programming languages is that the user has to specify step-by-step procedures the computer must carry out to obtain the output the user wants, using the input data.

It is true that the specification of these procedures has been made easier and easier through the years—assembly language made the task easier than machine coding by allowing the use of symbolic names instead of numeric code. High-level languages made the task even easier by allowing almost-English instructions instead of the almost-machine instructions of assembly language. For example, to add two numbers in assembly language, you have to give instructions like:

- Move first number from memory to register
- Move second number from memory to register
- Add the two numbers
- Move the result back to memory.

High-level languages allow the use of instructions like *Result := First + Second*, which certainly has made the writing of programs closer to human language. But it is still necessary to keep in mind the mode of operation of the machine and tailor the programs to these modes of operation.

The goal of programming in logic is to make the task of programming even easier. The user only has to focus on the relationships between the data available and the results required, leaving it to the PROLOG system to work out the procedure or steps required to obtain the results.

This characteristic of PROLOG can be illustrated by looking at the important facets of programming in PROLOG. The first facet is that the user supplies the system with a data base of facts relevant to the problem.

A fact in PROLOG is expressed as a relationship connecting one or more objects. For example, a PROLOG data base could consist of the following facts:

```
firstname(smith,john)
firstname(adams,abigail)
firstname(fonda,jane)
firstname (reagan,ronald)
firstname(vanburen,abigail)
firstname(kennedy,john)
```

The relationship is called the predicate (all the facts given above have *firstname* as the predicate), and the objects related by the predicate are the arguments. A predicate can have one or more arguments, though when there is only one argument, you tend to think of it more as a property than a relationship— for example, *young(jim)* or *sleeps(bob)*. Micro-PROLOG has a minor difference: commas are not necessary between objects. Also, with facts that involve one or two objects, you are allowed to use alternative versions, like *(bob sleeps)* or *(reagan first-name ronald)*.

The simplest use of PROLOG is to make a straight search to see whether or not a particular fact is in the data base. The various implementations of PROLOG make this search in slightly different ways. In the standard C & M versions, you would ask a question:

```
?-firstname(reagan,ronald).
```

In the micro-PROLOG version, you would ask:

```
is(firstname(reagan ronald))
```

In either case, the system would respond with "yes." The question *?-firstname(nixon,richard).* would produce the response "no" since the fact is not in the data base, even though it is true in real life. Asking a question is referred to as setting a goal for the PROLOG system to satisfy.

PROLOG can do much more than a straight search. The next step up would be to make a selective search by giving part of the information and asking the system to locate and supply the rest of the information. For instance, in Micro-PROLOG you could ask the question:

```
which( x : firstname(fonda x) )
```

and the system would search the data base and supply the answer:

```
jane
```

If a version close to the C & M standard is used, the question could be phrased:

```
?-firstname(fonda,What).
```

and the system would respond with:

```
What = jane
```

Important to note is that PROLOG works by pattern matching to try to satisfy the goal. Also, the symbols *X,Y,Z,x,y,z* . . . are treated by Micro-PROLOG as variables to which any value can be attached. Similarly, in the standard versions any word starting with an upper-case letter (*What* in the above example) is treated as a variable. So the system is able to match the question *firstname(fonda x)* or *firstname(fonda,What).* with the fact *firstname(fonda jane)* or *first-name(fonda,jane)* in the data base and supply the answer *jane* or *What = jane*.

What if more than one solution can be found by pattern matching? It is instructive to consider what would happen if you were to ask the Micro-PROLOG question:

```
which( y x : firstname(y x) )
```

PROLOG always works by searching the data base in the order the facts have been entered. So in this case the first match that would be found would be with *y= smith* and *x= john*. Micro-PROLOG would print out the solution:

```
smith john
```

then look for the next match and print out *adams abigail* and so on until it can find no more solutions.

Micro-PROLOG also allows you to get only as many of the possible solutions as you want by asking the question in the form:

```
one( y x : firstname(y x) )
```

In this case, Micro-PROLOG would print out the first solution it finds and then ask the user for a yes/no to whether more solutions should be searched for. Similarly, in the standard versions the system would produce one solution and wait for the user to indicate whether more solutions are to be searched for.

With the C & M standard, the user types in a semicolon and the system will search for alternative solutions; if the user hits the carriage return, the system will not look for any more possible answers.

One of the versions available for IBM PCs, PROLOG-86, uses a different convention. It automatically prints out all the answers. You can indicate to the system that you do not want more than one solution by using the cut operator, which is *!*. For example, you could ask a question as:

```
?-firstname(Who,john), !.
```

to indicate that you want only one solution. If the *!* is omitted, the PROLOG-86 system would find all possible solutions.

Rules in PROLOG

We have seen that the basis of PROLOG is its flexible, deductive data base. But what really lends PROLOG its power as a user-friendly language is that you can ask the system to draw logical deductions based on rules. All the user has to do is specify the pattern of deductions required, and the system takes on the task of interpreting the logical specifications as procedures

```
samefirstname(x1 x2) if firstname(x1 x) and firstname(x2 x) and not(x1 EQ x2)
samefirstname(One,Other) :- firstname(One,Something),
                            firstname(Other,Something),
                            One <> Other.
                       (standard version)
```

Listing 1.

and then executes them. In other words, a human being focuses on the logical specification of the problem while the PROLOG system attends to the procedural task of executing the specification.

To illustrate how rules work, let us consider the data base of first names we looked at before. Suppose the user wants to extract information on which pairs of persons have the same first name. A rule would be added to the data base (Listing 1).

You can see the Micro-PROLOG rule is really a logical specification saying that *x1* and *x2* have the same *firstname* if the *firstname* of *x1* and *x2* are the same *x*, and *x1* and *x2* are different persons (*not(x1 EQ x2)*). The other version of the rule says the same thing. *:-* stands for *if*, and < > stands for not equal to, though some versions use different symbols for these concepts. The user does not have to give any procedural directions to the system on how to implement this search task. It is the system that makes a procedural interpretation of the logical specification or declaration made by the user. It interprets this procedurally in the following manner.

To find two persons with the same first name, first find a person with some first name (*x* or *Something*), then find another person with the same first name and check that the two persons are not the same. It is useful to think of a rule as something that sets multiple goals for the system to satisfy.

In this example, if you ask the Micro-PROLOG question:

```
one( x y : samefirstname(x y) )
```

the system would print out the answer:

```
smith kennedy
```

and ask the user if another solution should be found. If the user says yes, the system will resume the search and give a second solution:

```
adams vanburen.
```

Let us analyze in detail how the system would find the solution to the standard PROLOG question:

```
?-samefirstname(Who,Whoelse).
```

It first tries to match the goal set by the question with a fact or rule in the data base. In this case, it can only find a match with a rule and matches *Who* in the question with *One* in the rule and *Whoelse* in the question with *Other* in the rule. A rule, as we have seen, is really a conditional goal connected by an *if* (or *:-*) with other goals. So here it tries to satisfy each of the goals in the rule, going from left to right.

The system is able to satisfy the first goal, *firstname(One,Something)* by matching *One* with *smith* and *Something* with *john.*

It now tries to satisfy the second goal, which has become *firstname(Other,john)*. It is important to note that in satisfying the first goal, the variable *Something* has acquired the value *john*—it is no longer a variable. The value *john* carries through to all the other goals in the rule.

The system satisfies the second goal by again searching the data base, starting at the top, and assigns the value *smith* to *Other*. Remember, to satisfy any goal, the system always starts its search at the top of the data base.

It now tries to satisfy the third goal, *One* < > *Other*. It fails to satisfy this goal because both *One* and *Other* have been assigned the value *smith.*

We now come to a very important characteristic of PROLOG: backtracking. When a goal fails, the system goes back to where it satisfied a previous goal and starts searching further in the data base. So in this case it cancels the value it allotted to *Other* and searches further down the data base to try and find a solution to the second goal, *firstname(Other,john)*. Clearly, it can satisfy this only when it gets to *firstname(kennedy,john)*. Now it again tries to satisfy the third goal, *One* < > *Other*, and succeeds this time since *One* is *smith* and *Other* is *kennedy*. Since all the goals in the rule have been satisfied, it prints out the solution:

```
Who = smith  Whoelse = kennedy
```

By applying this principle of backtracking, we can see that if we ask for more solutions, the system will backtrack to the first goal in the rule and satisfy *firstname(One,Something)* with *One* = *adams* and *Something* = *abigail.* It will now try to satisfy the second goal in the form *firstname(Other,abigail)*. Once again it will satisfy this second goal with *Other* = *adams*, but this will lead to failure on the third goal. So it will backtrack, find *Other* = *vanburen* to satisfy the second goal, and finally succeed on the third goal as well and print out the second solution *Who* = *adams* and *Whoelse* = *vanburen.*

Keeping in mind the fundamental principle of backtracking when satisfying multiple goals, the reader will certainly be able to deduce that if we ask for more solutions, the system will provide the answers:

```
vanburen adams
kennedy smith
```

in that order. In the standard version you would get a series of answers:

```
Who = vanburen  Whoelse = adams
```

and

```
Who = kennedy  Whoelse = smith
```

provided the system is prompted for more answers by typing a ; each time.

Let us consider one more example with an extended data base containing two more sets of facts (Listing 2).

A rule can be based on more than one type of fact. For example, suppose we want the system to use the data base to find out for us the names of owners of a particular type of computer in a particular city. That is, we want to ask for names of say, Apple II owners in Philadelphia. We could of course ask a combined question:

```
?-lives_at(Who,Somewhere,
     philadelphia), has_computer
     (Who,appleII).
```

What we are doing here is asking the

system to satisfy two goals or find matches for both the facts in our question. But it would be more convenient to design a rule as in Listing 3.

Now if we ask a question:

```
?-owns(Who,appleII,philadelphia).
```

the reply would be:

```
Who = adams
```

This response is in accordance with the principles we have discussed of the way PROLOG tries to find matches starting at the top of the data base.

We can see something of the flexibility of PROLOG by considering a different type of information we could extract from the same data base of rules and facts. If we pose the question in the form:

```
?-owns(Who,appleII,Where).
```

we would be able to get the names of all Apple II owners in our data base, along with the city in which they live.

Recursion in PROLOG

We have discussed the basics of how PROLOG can be used flexibly as a powerful, deductive data base. One other valuable feature of PROLOG is the facility it provides for writing recursive programs.

A recursive procedure or rule is defined as one that involves using the same rule as part of the application of the rule. A typical recursive definition would be for reversing a list of names or numbers in the following manner: to reverse a list, reverse the rest of the list, then add the first item at the end. This example shows that to reverse a list, we use the same reverse operation on the rest of the list.

What distinguishes a recursive definition from a circular one is that it does not go on endlessly. In the preceding example, when we apply the definition of reverse to the rest of the list (from item 2 onward), the definition tells us to again reverse the list from item 3 onward and put item 2 at the end of the list, which in turn leads us to reverse the list from item 4 onward. However, a stage is reached when the list is empty, and to reverse an empty list we keep it as it is. So the procedure does operate noncircularly (Listing 4).

The situation where recursion ends is called the boundary condition. In this case, it is the rule that reversing an empty list gives an empty list.

The recursive rule for finding the largest number in a list of numbers can be formulated as follows: to find the largest number in a list, compare the first number with the largest number in the rest of the list and choose whichever is bigger.

We can see that this involves finding the largest number in the list from item 2 onward, which in turn involves finding the largest number in the list from item 3 onward. But it is not endless, for the boundary condition is reached when the list has only one number—the largest number in the list is just that number.

To write a PROLOG rule for this we need a supporting rule for comparing two numbers, and designing this will illustrate some other features of PROLOG. Let us examine the rule:

```
greater(First,Second,Bigger):- First
        > Second,
Bigger is First.
greater(First,Second,Bigger):- Bigger
        is Second.
```

As we saw in previous examples, a rule involves trying to match conditional goals. In this case, to get a match if we type in a question, say:

```
?-greater(7,3,Bigger).
```

PROLOG will match *First* with 7 and *Second* with 3. To get a match for *Bigger*, it will try to match the conditional goals in the body of the rule following the :- symbol. In this case, since *First* (7) is greater than *Second* (3), it is able to match the goal *First Second* and decides that *First is Bigger*. The output will be:

```
Bigger=7.
```

To see how the second goal of the rule is used, let us analyze what happens if we ask:

```
?-greater(4,9,Bigger)
```

Arguing as before, we see that *First* and *Second* in the head of the rule would be matched with 4 and 9, respectively. But the goal *First > Second* does not succeed. Since the first rule fails, the alternative goal, *Bigger is Second* is satisfied and the result:

```
Bigger=9
```

is output.

To summarize, this pair of rules can be expressed in a free translation as: if *First > Second*, then give *Bigger* the value of *First*, else give *Bigger* the value of *Sec-*

```
lives_at (smith,111_elm_st, anytown).
lives_at (adams,3300_elfreths_alley, philadelphia).
lives_at (fonda,10_some_ave,los_angeles).
lives_at (reagan,1600_pennsylvania_ave,washington).
lives_at (vanburen,212_loop_way,chicago).
lives_at (kennedy,2010_presedential_blvd,boston).
and:
has_computer (smith, appleII).
has_computer (fonda, ibm_pc).
has_computer (reagan, ibm_pc).
has_computer (vanburen, appleII).
has_computer ( kennedy, appleII).
has_computer (adams,appleII).
```

Listing 2.

ond. We can see that this construct is analogous to the *IF* . . . *ELSE* of other languages.

We can now use this rule for selecting the greater of two numbers in a recursive definition for finding the largest number in a list of numbers. But first it should be noted that the representation of a list of items in PROLOG is by enclosing them in brackets with the items separated by commas—for example, the list of five letters, *a* through *e*, would be represented as *[a,b,c,d,e]*.

Second, PROLOG provides a notation distinguishing the first element of the list or head of the list from the rest of the list or tail, as follows: *[Head | Tail]*.

In this notation, *Head* = *a* and *Tail* = *[b,c,d,e]*. In other words, the head is the first element of the list, while the tail is a list consisting of all elements except the first. *Head* and *Tail* can, of course, be replaced with any variable names we prefer.

Now let us consider the recursive program for finding the largest number in a list. It consists of a pair of rules:

```
largest([Only],Only).
largest([First | Rest],Biggest) :-
            largest(Rest,Y),
        greater(First,Y,Biggest).
```

What happens if we ask a question:

```
?-largest([19],Result).
```

We see that PROLOG will be able to match the question with *largest ([Only],Only)* by identifying 19 with *Only* and *Result* with *Only* and so will output:

```
Result=19
```

But suppose we ask the question:

```
?-largest([3,6,7,11,4,2],Result)
```

Since PROLOG searches the data base from top to bottom, it will try the first rule with *largest* but will be unable to find a match because the rule has a list with a single element, *Only,* whereas the question has a list with six numbers in it. So PROLOG will now look for any other fact

```
  owns(Person,ComputerType,City) :-
                    lives_at (Person,Somewhere,City),
        has_computer (Person,ComputerType).
```

Listing 3.

```
        reverse of (a,b,c,d,e) is
 reverse of (b,c,d,e)                        followed by a
which is
reverse of (c,d,e)                   followed by b
which is
reverse of (d,e)               followed by c
which is
reverse of (e)        followed by d
which is
reverse of ()  followed by e
which is
 ()
```

Listing 4.

TURBO Power

PROGRAMMER'S UTILITIES

especially for Turbo Pascal on IBM PC/XT/AT and compatibles

MORE POWERFUL THAN UNIX UTILITIES!!!

Whether you are a
-- Student -- Hobbyist -- Professional Software Developer --
THESE UTILITIES WILL IMPROVE YOUR PROGRAMMING PRODUCTIVITY!!!

These Powerful, Ready-to-Use programs fully support Turbo Pascal versions 2.0 and 3.0, and MSDOS 2.X and 3.0. Here's what you get:

Pretty Printer
Standardize capitalization, indentation, and spacing of source code. Don't waste your own time! Several adjustable parameters to suit your tastes (works with any standard Pascal source).

Program Structure Analyzer
Find subtle problems the compiler doesn't: uninitialized and unused variables, modified value parameters, "sneaky" variable modification, redefined standard identifiers. Also generates a complete variable cross reference and a program hierarchy diagram. Interactive or write to file (works with any standard Pascal source).

Execution Timer
Obtain a summary of time spent in each procedure and function of your program, accurate to within 200 microseconds. Also counts number of calls to each subprogram. Fully automatic.

Execution Profiler
Obtain a graphic profile of where your program spends its time. Interactive, easy-to-use. Identify weak code at the instruction level. (Profiler and Timer for Turbo Pascal Source code only.)

Command Repeater
Go beyond MSDOS batch files to combine a powerful text parser with general-purpose command execution capability. Use to copy, print or delete across subdirectories, "make" programs and more.

Pattern Replacer
Find and REPLACE versatile regular expression patterns in any text file. Supports nesting, alternation, tagged words and more. Over a dozen programmer's applications included.

Difference Finder
Find differences between two text files, and optionally create an EDLIN script which rebuilds one from the other. Disregard white space, case, arbitrary characters and Pascal comments if desired.

Super Directory
Replace PCDOS DIR command with extended pattern matching, sort capability, hidden file display, date filtering, and more.

File Finder
Locate files anywhere in the subdirectory tree and access them with a single keystroke. Display the subdirectory tree graphically.

AVAILABLE IN SOURCE AND EXECUTABLE FORMAT

Executable: $55 COMPLETE including tax and shipping. Compiled and ready to run, includes user manual, reference card and one 5¼" DSDD disk. Ideal for programmers not using Turbo.

Source: $95 COMPLETE including tax and shipping. Includes all of the above, and two additional DSDD disks. Disks include complete Turbo Pascal source code, detailed programmer's manual (on disk) and several bonus utilities. Requires Turbo Pascal 2.0 or 3.0.

Requirements: MSDOS 2.X or 3.0, 192K RAM — programs run in less RAM with reduced capacity. Two drives or hard disk recommended.

TO ORDER:
VISA/MasterCard orders, call 7 days toll-free 1-800-538-8157 x830. In California, call 1-800-672-3470 x830 any day. Or mail check/money order to:

TurboPower Software
478 W. Hamilton Ave., Suite 196
Campbell, CA 95008

CIRCLE 45 ON READER SERVICE CARD

or rule with which a match can be found.

When it tries the second rule for *largest* it will match 3 with *First* and *[6,7,11,4,2]* with *Rest*. This will lead it to try to satisfy the goal *largest([6,7,11,4,2],Y)* in the body of the rule. To satisfy this, once again it has to use the second rule, which leads it to try to satisfy *largest ([7,11,4,2],Y)* . . . until it reaches a stage where it has to satisfy *largest([2],Y)*.

At this stage we see that PROLOG can use the first rule and identify *Y* with 2. It tried the goal *largest([2],Y)* when it tried to satisfy the head goal, *largest ([4,2],Biggest)*, at which stage it took 4 to be *First*. So it has now to go on to try to satisfy the second goal, *greater(4,2,Biggest)*, which it does, and to extract 4 as *Biggest* because of the operation of the rule for greater, which we analyzed earlier.

But the goal *largest([4,2],Y)* itself arose as the body goal when earlier trying to satisfy the head goal, *largest ([11,4,2],Biggest)*, so we see that the 4 has to be compared with the 11. Working backward this way, the 11 extracted by comparing 4 with 11 will in turn be compared with 7, until finally it outputs:

```
Result = 11
```

A convenient way of visualizing this recursive process is to think that the program first identifies 3 with *First*, puts it away somewhere, and then proceeds to deal with *largest([6,11,7,4,2],Y)*. But this in turn requires it to put 6 away somewhere and deal with the rest of the list.

So we may think of the 3,6, . . . as being successively stacked one on top of the other so that the program can later take them off the stack in reverse order for using them in the comparisons required in satisfying the second goal of *greater(4,2,Biggest)*, which leads to *greater(4,11,Biggest)*, which leads to *greater(11,7,Biggest)*, which leads to *greater(11,6,Biggest)*, which finally leads to *greater(11,3,Biggest)*.

Writing expert systems

This article has attempted to highlight some of the features of PROLOG and in particular emphasize its declarative or assertional way of writing programs as contrasted with the procedural methods that characterize other high-level languages. The declarative aspect of PROLOG makes the language a really handy tool in writing expert systems programs.

Expert systems are programs that emulate the working of a human expert. The approach that has been found most useful is to make a distinction between the knowledge base used by the expert and the inference program for drawing conclusions using the store of knowledge.

PROLOG lends itself ideally to storing the relevant knowledge in the form of facts and rules and then using the PROLOG interpreter to draw conclusions. Space considerations prevent enlarging on this use of PROLOG with examples. But depending on reader interest as communicated to the editor, we hope to have an opportunity in a subsequent article to illustrate the use of PROLOG for building expert systems. ■

References

Clocksin, W.F., and C.S. Mellish. *Programming in PROLOG.* New York: Springer-Verlag, 1981.

Clark, K.L. and McCabe, F.G. *Micro-PROLOG.* New Jersey: Prentice-Hall, 1984.

Bharath, Ramachandran. *Programming in PROLOG.* (TAB Professional & Reference Books, forthcoming by end 1985.)

Bharath, R. and Deb, A. *An Introduction to PROLOG: A Tutorial.* Proceedings of the National Meeting of the American Institute for Decision Sciences, Nov. 1984. (This article is a revised and expanded version of the tutorial.)

Ramachandran Bharath and Margaret Sklar are professors in the Management, Marketing and Computer Information Systems Dept. of Northern Michigan Univ.'s School of Business, Marquette, Mich. 49855.

TURBO EDITASM

Introducing the first co-resident editor assembler for the IBM PC family. ***TURBO EDITASM (TASM)*** is significantly faster and easier to use than the IBM Macro-Assembler (MASM). Whether you are new to assembly language and want to quickly write a small assembly language routine, or are an experienced MASM user tired of waiting months to assemble large files, ***TURBO EDITASM*** will bring the excitement back to assembly language.

***TURBO EDITASM* IS MUCH FASTER:**

- How fast is ***TASM?*** The graph below shows relative assembly times for a 48K source file. For large files like this we blow MASM's doors off at 3 times their speed. For smaller 8K files we positively vaporize them at 6 times their speed.

TASM	(110 sec.)
MASM	(340 sec.)

- ***TURBO EDITASM*** is faster for the following reasons: **(1)** Written entirely in assembly language (unlike MASM). **(2)** Editor, assembler and source file always in memory so you can go instantly from editing to assembling and back. **(3)** Eliminates the time needed to LINK programs. Executable COM files can be created directly. (Also creates OBJ files compatible with the IBM linker).

***TURBO EDITASM* IS EASIER TO USE:**

TASM includes many other features to make your programming simpler.

- Listings are sent directly to screen or printer. Assemblies can be single stepped and examined without having to leave the editor.
- Access the built-in cross reference utility from the editor.
- Full support of 186 and 286 (real mode) instructions.
- Both Microsoft and 8087 floating point formats are supported. 8087 and 287 instructions supported directly without macros for faster assembly.
- Calculator mode: Do math in any radix even using symbols from the symbol table.
- Direct to memory assembly feature lets you test execute your code from editor.
- Coming soon: A coordinated symbolic debugger.

COMPATIBILITY: *TASM* is source code compatible with MASM and supports macros, records and structures.

Introductory Price $49
With .OBJ Capability $99

Include $5.00 shipping and handling. California residents add 6% Sales Tax.

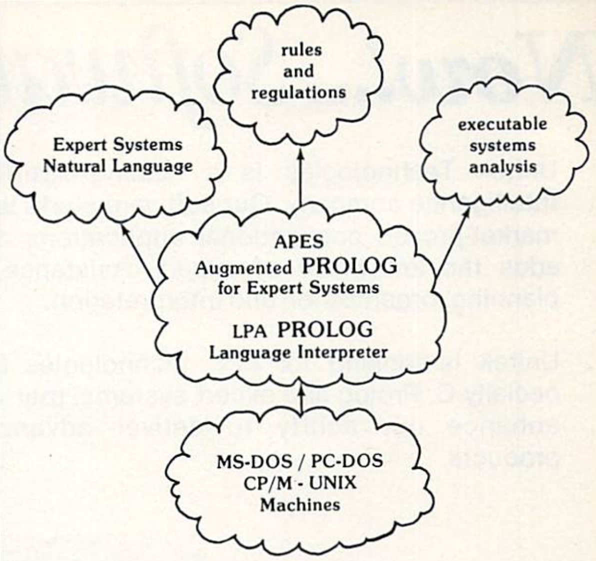
Dealer inquires welcome

916-988-7426
118 Buck Circle, Box C
Sacramento, CA 95630

Speedware™

IBM, Microsoft trademarks of IBM Corp., Microsoft Corp.

CIRCLE 41 ON READER SERVICE CARD


rules and regulations
Expert Systems Natural Language
executable systems analysis
APES
Augmented PROLOG for Expert Systems
LPA PROLOG
Language Interpreter
MS-DOS / PC-DOS
CP/M - UNIX
Machines

LPA PROLOG Language Interpreter 4.0 - MS-DOS	$395.00
LPA Sigma-PROLOG UNIX Version — 68,000	$695.00
LPA Sigma-PROLOG UNIX Version — VAX	$1500.00
LPA PROLOG Language Interpreter Documentation	45.00
APES Augmented PROLOG — MS-DOS	$395.00
APES for UNIX Systems	$695.00/$1500.00
APES Documentation	10.00
Pkg. LPA PROLOG & APES — MS-DOS	$650.00

PROGRAMMING LOGIC SYSTEMS
31 Crescent Drive
Milford, Connecticut 06460
(203) 877-7988

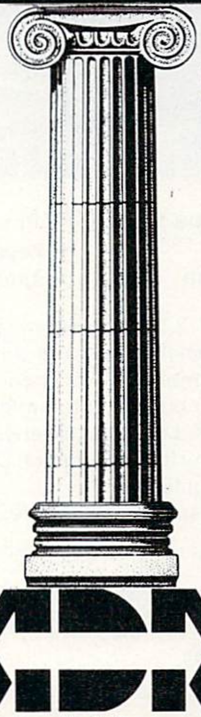
CP/M is a trademark of Digital Research Inc.
MS-DOS is a registered trademark of Microsoft Corp.
PC-DOS is a registered trademark of IBM
UNIX is a trademark of Bell Labs.

Educational & Pkg.
Discounts available
Prices subject to change

CIRCLE 21 ON READER SERVICE CARD

THE SYMBOLIC *SINE QUA NON*
L I S P

THE fifth generation language
P R O L O G



Cybermetrics **UNXLISP-86** (tm) Features:

- 1 Megabyte address space.
- LAMBDA, NLAMBDA, MACRO with displacement.
- CATCH, THROW, ERRSET, & ERR.
- SAVE/RESTORE virtual images to/from disk.
- &OPTIONAL, &REST, and &AUX parameters.
- Tree-structured object list, functional directories.
- Lisp utilities with source, including pretty-printer, structure editor, debugging functions.
- Works with any text editor through exec function.
- Extensive manual – written in ENGLISH
- Requires 320K memory and runs under MSDOS and PCDOS versions 2.0 and later.

TECHNICAL
(408) 725-1344

$49.00

ADD $3.00 FOR
SHIPPING & HANDLING

Implementing the full Edinburgh Syntax as described by Clocksin and Mellish.

Recognized by Japan as providing unparalleled opportunity for artificial intelligence.

Applications:

- The highest level of a hierarchial robotic control system.
- Machine recognition of natural language.
- Expert systems and knowledge engineering.

Optional:

- Virtual memory
- Special libraries
- Language extensions
- Large model

Requires 192K memory and runs under MSDOS and PCDOS versions 2.0 and later.

TECHNICAL
(215) 646-4894

educational package
$29.95
other versions $50–$500

ADA
automata design assoc.

1570 ARRAN WAY
DRESHER, PA 19025
TO ORDER CALL **(215) 355-5400** USE IT FOR 30 DAYS

VISA
MASTERCARD

CIRCLE 1 ON READER SERVICE CARD

Now!... Software with know-how™

Unitek Technologies is a leading Artificial Intelligence company. Our software starts with market-proven conventional applications and adds the know-how of expert assistance in planning, organization and interpretation.

Unitek is looking for new technologies (especially C, Prolog and expert systems) that will enhance our ability to deliver advanced products.

We also want additional, qualified staff.

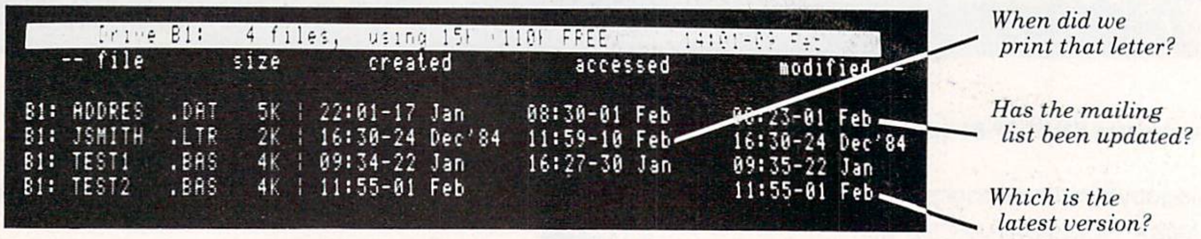
- If you would like to do state-of-the-art R&D in applied AI,
- If you are excited by the prospect of making AI commercial,
- or if you have developed a new AI technology, we encourage you to contact:

Dr. Vance Giboney, Unitek Technologies Corporation
#115-10751 Shellbridge Way, Richmond, B.C. Canada
V6X 2W8 (604) 276-2429

unitek™
TECHNOLOGIES CORPORATION

CIRCLE 23 ON READER SERVICE CARD

DATESTAMPER™ has the answers


When did we print that letter?
Has the mailing list been updated?
Which is the latest version?

```
   Drive B1:    4 files,  using 15K  (110K FREE)      14:01-09 Feb
   -- file          size        created            accessed            modified

B1: ADDRES   .DAT    5K | 22:01-17 Jan      08:30-01 Feb      08:23-01 Feb
B1: JSMITH   .LTR    2K | 16:30-24 Dec'84   11:59-10 Feb      16:30-24 Dec'84
B1: TEST1    .BAS    4K | 09:34-22 Jan      16:27-30 Jan      09:35-22 Jan
B1: TEST2    .BAS    4K | 11:55-01 Feb                        11:55-01 Feb
```

DateStamper™ keeps your CP/M computer up-to-date!

- avoid erasing the wrong file
- back-up files by date and time
- keep dated tax log of computer use
- simplify disk housekeeping chores

OPERATION: DateStamper extends CP/M 2.2 to automatically record date and time a file is created, read or modified. DateStamper reads the exact time from the real-time clock, if you have one; otherwise, it records the order in which you use files. Disks prepared for datestamping are fully compatible with standard CP/M.
INSTALLATION: Default (relative-clock) mode is automatic. Configurable for any real-time clock, with pre-assembled code supplied for all popular models. Loads automatically at power-on.
UTILITIES: Enhanced SuperDirectory • Powerful, all-function DATSWEEP file-management program with date and time tagging • Installation and configuration utilities
PERFORMANCE: Automatic. Efficient. Versatile. Compatible.

Requires CP/M 2.2. Uses less than 1K memory. Real-time clock is optional.

When ordering please specify format

8" SSSD, Kaypro, Osborne Formats $49
For other formats (sorry, no 96 TPI) add $5.
Shipping and handling $3
California residents add 6% sales tax
MasterCard and Visa accepted

Specialized versions of this and other software available for the Kaypro.
CP/M is a registered trademark of Digital Research, Inc.

Write or call for further information

Plu*Perfect Systems

BOX 1494 • IDYLLWILD, CA 92349 • 714-659-4432

CIRCLE 35 ON READER SERVICE CARD

Screen scroll utilities and a gnu

By Tim Parker

This month we will briefly detour from the planned scheme of things to answer some requests sent in by several readers.

Those who do a lot of listing on the screen with the MS-DOS *TYPE* command or who look at program listings in BASIC, for example, are well acquainted with the problem of listings zipping past on the screen too quickly, or worse, having the part of the listing that was of interest disappear off the top of the screen before the listing is frozen. Scroll Lock and the Control-S, Control-Q methods of pausing the display work well, but they lack the refinement that could make screen dumps very useful.

As many of us do screen dumps of documentation, listings, or multiple directories, it is time to take a look at a few programs designed to make life with the scrolling screen a little easier. While this may not seem like an important issue, it is actually a frequently requested item that rarely gets mentioned in examinations of the public domain libraries.

Vincent Bly has developed a program called Re-View. It is a user-supported program available on several sources, including PC-SIG vol. 138. A contribution of $15 is requested.

For those who didn't catch my earlier remarks on user-supported programs, a quick digression. User-supported programs are intended for free distribution and are usually available through bulletin boards, public domain libraries, user groups, and other sources. The authors of the programs essentially release the program to the public domain but add a rider saying they would appreciate a given amount of money (usually $15-$30) if the program is found useful. There is no obligation to send the requested amount, and there is nothing illegal about not doing so.

The advantages of sending the contribution are the support and the ensuing updates that are available quickly and readily. Also, the programmer will be encouraged in many cases to continue writing programs and rewriting versions of existing programs. This allows programs to be distributed widely and gives a user a free look at the material.

The user-support idea eliminates one of the major headaches of commercial software: buying a program for several tens or hundreds of dollars, only to find it doesn't do what the user thought it would. Further, the ever-present piracy and illegal duplication problems are much less important when the program is allowed to be distributed. True, the author may not make as much money with user-supported programs as if the program had been released in a commercial package, but most of the programs released in the user-support fashion are not the programs that can cost up to several hundred dollars.

Now for some details about Re-View. Re-View gives IBM and compatible machines a buffer that holds 75 lines in it. As text is scrolled off the top of the display, it is loaded into the buffer and can be recalled easily. With the buffer and the current screen together, this gives a 100-line "virtual" screen that is fully accessible. The lines can be recalled either one at a time or a page at a time.

Re-View requires a color/graphics board. It does not work with the IBM monochrome display adapter card, but it works fine with Compaqs and other computers that have a dual purpose card.

Re-View is activated by simply typing the program's name. (You may wish to include it in an AUTOEXEC.BAT file for convenience.) It will create the necessary buffer and activate the keys it uses for commands. These are the right-hand plus and minus keys by the numeric keypad. The standard plus and minus keys are unchanged. The program is temporarily deactivated by using a Control-ALT/Scroll Lock sequence. This will be necessary whenever the program is resident in memory and another program is used which also uses the plus or minus keys. Re-View also will not work with programs that access screen memory directly or use pages other than page 0 for the screen.

The plus key scrolls through the screen buffer one line at a time toward the top, while the minus key scrolls line by line toward the bottom. A Shift-plus combination goes up by pages (25 lines), while a Shift-minus goes down by pages. The scrolling is fairly smooth compared to the usual color/graphics scroll, but when compared to the dual function boards available or a Compaq which supports both monitors, it seems a bit jerky.

The saving to the buffer can be toggled on or off with a ALT/Scroll Lock combination. The status is fed back to the user by beeps from the speaker. Two beeps signify that the save is activated; one beep means it is deactivated. This is useful when an introductory or reference section is loaded into the buffer and recalled for quick referral when needed while scrolling through the rest of the text. Alternatively, one text can be stored in the buffer and recalled at a later time, or when in another program, or while scrolling through another.

Although Re-View is most useful with the standard scrolling commands in DOS (*TYPE, DIR*, etc.), it can also be used inside other programs and languages, such as Microsoft's BASIC, that use a screen dump for some functions, such as listings. A long listing can be stored in the Re-View buffer and skipped through when jumps are encountered or variable references used. The buffer's contents are destroyed when some programs are run, though, and *SCREEN* and *WIDTH* commands clear the buffer.

Re-View will not work with the DOS Version 2.x ANSI.SYS program. This is because the ANSI.SYS program does not support the usual routines in ROM for scrolling. However, a patching program is supplied with Re-View that will correct the problems in the ANSI.SYS file, creating a new file called ANSIR.SYS (so the original is not destroyed). The new file is then called from the CONFIG.SYS file.

Two other programs are supplied with Re-View. CLA is a program that works like DOS's *CLS* command but which also clears the contents of the virtual screen buffer. UP adds the facility to scroll the text above the cursor into the buffer.

Re-View proved to be especially useful when disks were being scanned for contents and the directories saved in the buffer for reference. Further, when a long document file is examined, the recall features were used instead of having to print the file out. In this respect, Re-View would prove very useful for someone who does a lot of text processing.

Those who have used mainframes running VM/CMS will be acquainted with the command *BROWSE*. *BROWSE* lets the user scan a file and move through it to the top or bottom, left or right, just as if a word processor was invoked, but without the edit commands. A similar program is available on PC-SIG vol. 205.

When the command *BROWSE FILENAME* is entered, the screen is cleared and the text is rapidly displayed. The top two lines of the screen are reserved by *BROWSE* and are shown in reverse video. The file being scanned is named in the upper left corner, while a running line and column count appear in the upper right corner in the format "Line 73 of 156 Column 30 of 120".

The cursor movement keys are used to move through the file. The up and down arrows move through a page at a time, while the left and right arrows move the text to the left and right, with some overlap for readability. The Home key returns to the top left of the file, while the End key goes to the bottom left.

The *BROWSE* function can be exited using the ESC key. *BROWSE* is useful in that it is fast and allows movement anywhere in the document or program listing being examined. Although it cannot operate over another program like Re-View, it is useful in its own right and will probably replace the DOS *LIST* command for many people once they have tried it.

Another program that acts like Re-View but adds other features is available on PC-SIG vol. 198 under the name L4. Written by Vernon Buerg, it runs only on DOS 2.x and will not use ANSI.SYS.

L4 is invoked exactly like LIST, but allows the user to issue a number of positioning commands. These are all relative to the cursor's current position and will affect the block of text in memory. This is variable depending on the amount of free memory in the machine, to a maximum of 64K.

The name of the file is displayed in the top left of the screen. The bottom line is reserved for commands, and a quick command index appears in the lower right corner. The file is displayed in half intensity.

Commands can be issued by either letters or control keys. Files are scanned with the normal arrow keys: up and down arrows move up or down one line, while the left and right arrows move 20 columns to the left or right. Alternatively, the letter keys N (for next) and P (for previous) move one line up or down, while the L (left) and R (right) keys move 20 columns to the left or right, respectively.

Paging through a document is done with the PgUp and PgDn control keys or with the D and U keys. Striking the RETURN key advances you to the next page. The Home key (or T, for top) moves you to the start of the file, and the End key (or B, for bottom) moves you to the end. The ESC key exits, as does either Q (quit) or X (exit). Striking the F1, H, or ? keys will reveal the command list.

L4 also has a text search facility, which is not found in any of the previously mentioned programs. A character string can be searched for by issuing the slash (/) command, followed by the text to be located. The maximum text length is 32 characters. The string to be located is then displayed on the command line and the text in the memory block is scanned. If a match is found, the line blinks. If no match is found, an error message is issued. To scan another block of memory, the PgUp or PgDn command must be used to load new text into the memory block and the F3 key used to reissue the scan command. F3 is also used to locate the next occurrence of a match in the current memory block, if required.

A few other commands that move within the memory block are also provided. These commands also move within the entire text when the find function is used. Control-Home restarts from the current memory block, while Control-PgUp goes to the start of the file. Control-PgDn skips to the end of the file, and Control-left arrow resets the scrolling to column one. The find facility is exited with the F10 key.

L4 takes longer to load than either of the other two programs mentioned in this

---

Instant-C™: The Best Value In C Programming Tools

The edit-compile-link-test-debug cycle that takes tens of minutes with compilers and linkers is only seconds with the *Instant-C* interpreter. Yet it runs your programs **50 to 500 times faster** than conventional C interpreters! You get the best of both compilers **and** interpreters. Only *Instant-C* is a complete, integrated environment for creating, testing, and running your programs.

Instant-C gives you **all** of these proven capabilities in one tightly integrated package:

interpreter—*Instant-C* runs your programs faster than some compilers; has direct execution; full K&R

compiler—*Instant-C* can make stand-alone programs

full-screen language editor—shows syntax errors with cursor set to trouble spot

C source debugger—single-step, breakpoints, stack trace, more

run-time checker—validates pointer refs, array bounds, more

C source formatter—save editing time, find logical flaws

standard library with source—for best portability

linker—work with multiple source modules

Lint—extensive compile-time validation

The cheapest available examples of these tools would cost $800 (and they don't even work together). You could spend close to $3000 to get the best product of each kind, but you'd have ten times the complexity, filling megabytes of disk. *Instant-C* is faster: it performs these functions automatically. *Instant-C* is far more than the sum of its parts.

Instant-C is all of these capabilities in one package, fits on a single floppy disk, is full K&R, works on IBM PC's, compatibles, and others under DOS or CP/M-86. It costs only $495.

Instant-C is the best value in C programming tools. Guaranteed, or your money back for any reason in first 31 days.

Rational Systems, Inc.

(617) 653-6194
P.O. Box 480
Natick, MA 01760

Instant-C is a trademark of Rational Systems, Inc.

CIRCLE 72 ON READER SERVICE CARD

article, but it has the text find facility, if it is required.

Of the three *TYPE* replacements, one is probably of greater use to a programmer or writer, while another works well for a casual DOS user. Experimentation ultimately determines which is best, if indeed a *TYPE* replacement is deemed necessary.

Although a gnu is defined in the dictionary as "an African antelope with an ox-like head and a long tail" or a "wildebeest," the GNU in this month's column has nothing to do with Africa. The "GNU Project" has achieved something of a cult status in the computer world. Many people have heard about it but few really know what it is and even fewer have up-to-date information. The head of the project, Richard Stallman, has recently tried to rectify this problem by placing updates on services such as CompuServe, bulletin boards, and in some magazines.

GNU is an attempt to create a free version of UNIX. Its name stands for "GNU's Not UNIX," which is an excellent example of a recursive name if ever there was one! The intention is to create a complete system that is UNIX compatible in every sense but which lacks the exorbitant costs associated with UNIX. If eventually completed, GNU will be distributed free of charge and modifications and redistribution will be allowed, but no one will be able to charge for GNU or a modified version.

To date, Stallman reports that a portable implementation of both C and Pascal compilers are completed. An EMACS-like text editor, a yacc-compatible parser generator, a linker, and 30 to 40 utilities are ready. The crucial command interpreter, or shell, is supposedly near completion, with the kernel and debugger targeted for distribution by the end of this year.

The aims for GNU are high. Although the goal is to make it UNIX compatible, some changes will be implemented to aid in operating convenience. It is intended to have both C and LISP as primary languages, with longer file names, windowing, and terminal independent display support.

At the moment, GNU is being implemented on 68000/16000-class machines with virtual memory. Despite many rumors to the contrary, the adaptation to other machines such as the IBM-PC is not intended to be part of the original GNU plan.

The fate of the GNU project, and whether it makes it down to the IBM PC, is still to be seen. It is an ambitious project, and although the goals are commendable, it is likely to generate some animosity from the industry. Stallman (who wrote the original EMACS editor) can be contacted through InterNet, or at 166 Prospect St., Cambridge, Mass. 02139.

Recently crossing my desk was a press release from the New York Amateur Computer Club Inc. (NYACC). NYACC, founded in 1976, is one of the oldest and largest organizations of microcomputer users in the world. Although well known for its support of 8080 and Z80 programs, it has recently unveiled its PC/Blue Library of IBM PC public domain material.

For information about this organization and the available material, address your letters to NYACC, Box 106, Church St. Station, New York, N.Y. 10008. A catalog containing a list of diskette contents, an alphabetical index of files, and a list of programs grouped into 19 topics is available for $5. At last check, there were 110 disks in their library. Most are double-sided. Although much of the material is contained in bulletin boards and from such other sources as PC-SIG (1556 Halford Ave., Suite 130, Santa Clara, Calif. 95051), NYACC disks are inexpensive at $7 each and provide a fairly large source of public domain material on the east coast.

Two of the programs mentioned in this column are available from NYACC: ReView on vol. 52, and L4 on vol. 81. (An earlier version of L4 can be found on vol. 68.)

Many volumes from NYACC match those of PC-SIG, especially user-supported programs. Having this large, trustworthy source of public domain material now available on both coasts of the United States is a definite benefit for all users. All the programs mentioned in this column will be available on the *COMPUTER LANGUAGE* Bulletin Board Service (BBS) and CompuServe.

Please continue to submit your comments, suggestions, and requests for programs to me in care of *COMPUTER LANGUAGE*. Reader feedback is the best judge of what is wanted out there, and so far you've been very helpful. Short notes also can be left on the BBS, but letters really are the better form, as I can be more sure of seeing and responding to each one.

REMOVE

from your **C** programs with

PC–LINT

PC–LINT analyzes your C programs (one or many modules) and uncovers glitches, bugs, quirks and inconsistencies. It will catch subtle errors before they catch you.

PC–LINT resembles the Lint that runs on the UNIX O.S. but with more features and greater sensitivity to the problems of the 8086 environment.

- Full K&R C
- Supports Multiple Modules—finds inconsistencies between declarations and use of functions and data across a set of modules comprising a program.
- Compares function arguments with the associated parameters and complains if there is a mismatch or too many or too few arguments.
- All warning and information messages may be turned on and off globally or locally (via command line and comments) so that messages can be tailored to your programming style.
- All command line information can be furnished indirectly via file(s) to automate testing.
- Use it to check existing programs, programs about to be exported or imported, as a preliminary to compilation, or *prior* to scaling up to a larger memory model.
- All one pass with an integrated preprocessor so it's very fast.
- Has numerous flags to support a wide variety of C's, memory models, and programming styles.
- **Introductory Price: $98.00 MC, VISA** *(Includes shipping and handling) PA residents add 6% sales tax. Outside USA add $10.00.*
- Runs on the IBM PC (or XT, AT or compatible) under DOS 2.0 and up, with a minimum of 128KB of memory. It will use all the memory available.

GIMPEL SOFTWARE

3207 Hogarth Lane • Collegeville, PA 19426

(215) 584-4261

*Trademarks: IBM *(IBM Corp.)*, PC–LINT *(Gimpel Software)*, UNIX *(AT&T)*

SuperSoft Languages When Performance Counts

A programmer's most important software tool is the language compiler or interpreter he uses. He has to depend on it to work and work well.

At SuperSoft, we believe it. That's why we offer four excellent compilers: SuperSoft FORTRAN, SuperSoft A, SuperSoft C, and SuperSoft BASIC. They answer the programmer's need for rock solid, dependable performance on microcomputers.

SuperSoft FORTRAN

With large code and data.
SuperSoft FORTRAN version 2.0 with large code and data space is now available under MS DOS and PC DOS. It gives you the power to compile extremely large FORTRAN programs on micros. It allows double precision and complex numbers, full IEEE floating point, and a full range of other important features for the serious FORTRAN programmer. Both 8087 support and a RATFOR preprocessor are optionally available.
FORTRAN (CP/M-80 & 86, MS DOS, PC DOS): $325
8087 support: $50 RATFOR: $100

SuperSoft A

A true Ada* subset
SuperSoft A is a completely standard subset of the Ada language, incorporating approximately 63% of the standard Ada syntax and including such important features as packages and separate compilation. For CP/M-80 microcomputers: $300.

SuperSoft C

SuperSoft C is a high-powered, full-featured C compiler designed for serious C applications. It is fast – both in compilation and execution, and it is packed with more than 200 library functions (all delivered in source code form). SuperSoft C produces optimized assembly code, and object code can be ROMed.

SuperSoft C (for CP/M-80, CP/M-86, MS DOS, PC DOS): $350

SuperSoft BASIC

The SuperSoft BASIC compiler lets you get serious with business and financial programs. It uses BCD math to give you highly accurate results for demanding applications. SuperSoft BASIC is a true native code compiler that is generally compatible with Microsoft's BASIC interpreter. And an additional bonus – no run time license fee is required.

SuperSoft BASIC Compiler (for MS DOS, PC DOS, and CP/M-86): $300

Also available for programmers:

Star-Edit, a full-featured programmer's text editor: $225.00
Disk-Edit, an invaluable programmer's disk data editor: $100.00

To order call: **800-762-6629**

In Illinois call 217-359-2112

SuperSoft

SuperSoft, Inc., 1713 S. Neil St., P.O. Box 1628, Champaign, IL 61820

In conjunction with SuperSoft, SuperSoft FORTRAN was developed by Small Systems Services, Urbana, IL, a leader in FORTRAN development.

Japanese Distributor: ASR Corporation International, TBL Building, 7th Floor, 1-19-9 Toranomon, Minato-Ku, Tokyo 105, Japan Tel. 03-5025550. Telex 222-5650 ASRTYO J.

*Ada is a trademark of the Department of Defense
PC DOS is a trademark of International Business Machines.
MS DOS is a trademark of Microsoft.
CP/M-80 and CP/M-86 are trademarks of Digital Research, Inc.

CIRCLE 74 ON READER SERVICE CARD

MRS: An experimental AI system

By John Sechrest and Nick Flann

In artificial intelligence programming, you generally try to reason about time, states, or alternatives. There are programming techniques that make this reasoning easier.

If you program in a traditional programming language, you end up building your own set of tools to support AI programming. LISP, for example, has become popular because it manages data structures easily and its programs can manipulate other programs. PROLOG has gained its status because it is a logic programming language that changes programming from a proscriptive activity to a descriptive one.

However, both of these languages have their shortcomings when it comes to AI programming. LISP has no tools for logic programming, and PROLOG has no efficient mechanism for closely controlling a procedural activity. Without rewriting parts of PROLOG, it is difficult for programs in PROLOG to reason about their own behavior.

MRS is an experimental logic programming system written in LISP by the Stanford University Heuristic Programming Project. The most significant difference between MRS and PROLOG is MRS's ability to observe and control its own activity.

MRS provides tools to manipulate meta-level information. With these tools, representations, inference methods, and search strategies can be switched. A LISP function also can be attached to a predicate to increase flexibility and efficiency as needed. This ability to control the meta-level structures of a problem makes MRS a useful tool for AI programming.

Syntax. MRS uses a prefix notation. This notation is inherited from the LISP in which MRS is written. In MRS, a set of primative symbols is used to represent things and facts about things. An object can be almost anything: Fred, car, hobbit. Relations describe the relationship between objects and are used to state facts. For example, the line:

```
(Uncle Bilbo Frodo)
```

can represent the statement that Bilbo is Frodo's uncle. Some of the symbols have predefined meanings, such as logical operators. All statements are collections of terms in a prefix notation.

All logic systems are merely syntactic manipulation systems. The semantics of a statement are independent of the syntax. As long as the statements are used consistently, the semantics of the statements will be maintained. Many bugs occur because people assume some semantic component to a statement about which MRS has no information and therefore no ability to reason about it.

Logical operators. A set of logical operators *(and, or, not, if)* is built into MRS. These operators allow the combination of simple facts to create more complex facts:

```
(and (hungry Bilbo) (tired Bilbo))
(if (tired Bilbo) (sleepy Bilbo))
```

These operators work in the obvious way: *and* succeeds if all of its subclauses succeed, *or* succeeds if any of its subclauses succeed, and *not* succeeds when the subclause doesn't.

If is the equivalent of the PROLOG implies. In PROLOG you might write:

```
P:- Q. (P is implied by Q)
```

In MRS you would say:

```
(if Q P)
```

Variables. If you wanted to refer to some undetermined object instead of specific objects, you would use a variable to refer to the object. There are two types of variables in MRS: base-level and meta-level. Base-level variables begin with a *$* and are used to refer to objects. Meta-level variables begin with an *&* and are used to refer to statements.

Now you can make statements like:

```
(if (wear magic-ring $x) (invisible $x))
(if (orc $x) (ugly $x))
```

Note that if you just say *(ugly $x)*, *$x* will match all objects in the whole world, in other words, "All things are ugly." To limit the scope of a statement, you must use an *if* clause.

Data base. MRS also provides functions to add and delete facts from its data base. To store a fact in the MRS data base, use *assert*:

```
(assert '(fat Bilbo))
```

Notice the single quotation mark character. *Assert* is a LISP function that takes LISP variables as arguments. If you are putting a constant into the data base, you must quote it to make it a literal. *Unassert* will remove a fact from the data base. Both *assert* and *unassert* can do additional inferencing when storing a fact.

The data base can be looked at two ways. You can look for items either stored in the data base or implied by items in the data base. If, for example, you say:

```
(assert '(age Bilbo 111))
(lookup '(age Bilbo $x))
```

MRS will return:

```
((|$x|. 111) (t.t))
```

This is a binding list describing the bindings of the variables. If you want to find an item implied by the data base rather than look into the data base for items stored there, you would use *truep*. *Truep* will do backward inferencing through the data base to find if a fact is implied by the data base.

Instead of looking at binding lists in all of the examples, let's define a function to print the lists in readable form. A LISP function called *output* prints a fact from the data base based on a template that you provide. Listing 1 shows some definitions that make function output look more reasonable.

If you say:

```
(find '(age bilbo $x))
```

MRS will return:

```
Bilbo is 111 years old
```

instead of

```
((|$x|.111)(t.t))
```

Truep also searches for the implications of a data base. If you look at the following set of statements, you can see that the fact that Bilbo is happy is not in the data base but is implied by it.

```
(assert '(if (hobbit $x) (happy $x))
(assert '(hobbit Bilbo))
(lookup '(happy $x)) ; this will return
    nil
(find '(happy $x))
Bilbo is happy.
```

Default inferencing. Most programs written in a logic language may not need all of the fancy facilities MRS provides. The default state for MRS is backward inferencing. Switches add forward inferencing, justification, caching, and agenda processing. In the default state, these are turned off because each adds to the overhead of the system.

Assert does forward inferencing, while *truep* does backward inferencing. Let me illustrate the difference. (Note that changing the inference method will not alter the answer but will alter the amount of work needed to find the answer. Whether the alteration is good or bad depends on the structure of the rules.) We can turn on forward inferencing for a set of statements:

```
(assert '(toassert (&x) fc))
```

for forward inferencing with the following rules:

```
(assert '(if (orc $x ) (ugly $x)))
(assert '(orc ragashak))
```

When you assert *(orc ragashak)* then *(ugly ragashak)* will be stored in the data base. Then the process of asking if ragashak is ugly is just a lookup. In other words, *(ugly ragashak)* is already in the data base.

If the forward inferencing on orc was not turned on, inferring the answer would take longer. When asked if ragashak is ugly [*(find '(ugly ragashak))*], the backward inferencing procedure looks for *(ugly ragashak)*. When this fails, it looks for rules like *(if (. . .) (ugly)*. MRS will find the rule *(if (orc $x) (ugly $x))* and substitute ragashak for *$x*. It will then look for *(orc ragashak)* and will succeed.

Rule structure determines which type of inferencing is more efficient. If the rules are forward branching, then backward inferencing is more efficient. If they are backward branching, then forward inferencing is more efficient.

Although I defined forward inferencing with *(toassert (&x) fc)*, I could just as easily have said *(toassert (orc &x) fc)*. This procedure causes only facts with the form *(orc &x)* to be forward-inferenced. All others will continue to be backward-inferenced. Note that if you use *$x* instead of *&x*, MRS will match only statements of the literal form *(orc $x)*. It will not find a statement of the form *(orc $y)*. That is the purpose of the meta-level variable: to allow you to reference statements that contain base-level variables.

The ability to specify the types of statements you do forward inferencing on allows you to tailor inferencing to the data. Suppose, for example, you have the following rules:

```
(assert '(if (orc $x ) (ugly $x)))
(assert '(if (orc $x ) (mean $x)))
(assert '(if (orc $x ) (big $x)))
(assert '(if (hobbit $x ) (nice $x)))
(assert '(if (dwarf $x ) (nice $x)))
(assert '(if (elf $x ) (nice $x)))
(assert '(orc ragashak))
(assert '(hobbit bilbo))
```

If you look closely, you will notice that rules about orcs are forward branching. The rules about things that are nice are backward branching. No matter what type of inferencing you choose, you still get the same answer. However, for some of the rules it is more efficient to do forward inferencing instead of backward inferencing. To find out *(ugly ragashak)*, you don't care if he is big and mean, so you want to do backward inferencing. To find out *(nice bilbo)*, you don't care if *(nice dwalin)*, so forward inferencing on hobbit will be more efficient.

As you can see, both inferencing methods will work, but one will take longer than the other. By using information about the structure of your data, you can make the inferencing more efficient.

Basic AI tools. As I stated earlier, justification, caching, and search control normally are turned off in MRS. All of these add overhead in time and space. But you should alter how the defaults are set if you need to follow the reasoning for a problem, if you know that a specific type of inference is going to be done repeatedly, or if you have a problem with a structure that is inappropriate for depth-first searching.

Justification. Using the previously cited rules, we can show how the reasoning is done. We can turn on justification by saying:

```
(setq justify 'justifications)
```

Then when we say *(find '(ugly $x))* all of the reasons that lead to this conclusion are stored into the justification theory. If we say:

```
(why '(ugly ragashak))
```

MRS will respond with:

```
p274: (ugly ragashak) by bc
    p272: (orc ragashak)
    p273: (if (orc $x) (ugly $x))
```

giving you all of the work that it went through. *p272:* is a reference to where the fact is stored.

Caching. Using the previously cited rules, if you say *(lookup '(ugly $x))* it will return nil. On the other hand, if you say *truep* a binding list will be returned. Even after *truep* has returned a binding, lookup will not find a reference. This is because the facts inferred in the process of inferring a statement are not saved. If every inference were saved, you would soon run out of space. But if you know that some inference will be used frequently, you can say:

```
(setq cache 'cache)
```

and all of the inferred facts will be stored in a theory called cache. Theories are just collections of facts stored under a name. You can switch or empty theories. This gives you control of the facts that are stored in the data base. Just think of them as buckets that facts are put into. You can remove all of the facts from a theory by typing *(empty 'cache)*.

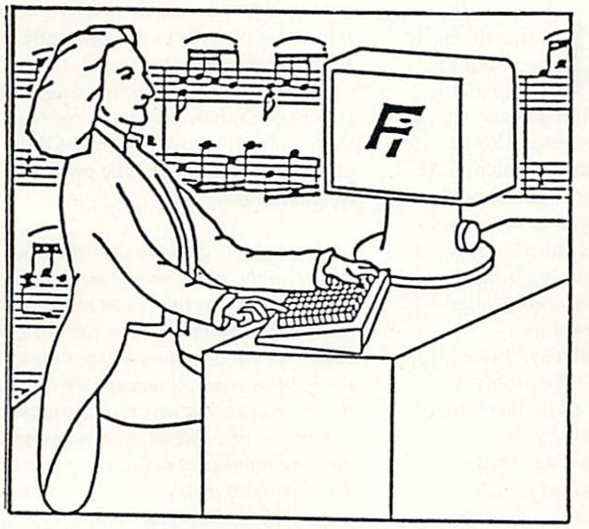
Search control. Since many reasoning tasks are done in very large spaces, you will need to manage not only the memory usage (with caching) but the time as well. Search control can reduce the amount of work a program may need to do. Both the base- and meta-level interpreters work by taking tasks from an agenda and executing them. The *preferred* command allows you to order the items on the agenda. Since this ability adds more overhead, the ordering feature is normally disabled. You can turn it on with *(setq preferred t)*.

After the *preferred* switch is set, you can make statements like the one presented in Listing 2. But this procedure

```
(assert '(template (&x &y) (&y is &x)))
(assert '(template (hobbit &y) (&y is a hobbit)))
(assert '(template (orc &y) (&y is a orc)))
(assert '(template (age &x &y) (&x is &y years old)))
(defun find (x) (output (plug x (truep x))))
```

Listing 1.

Franz *the first name in* **Lisp**



- Runs a wide range of applications and AI development environments including OPS-5, Flavors, MRS, GLISP, Pearl.
- Compiler, interpreter and library of LISP functions.
- The most widely distributed dialect of LISP.
- Common LISP compatibility.
- Can access C, Fortran and UNIX shell from within a LISP program.
- Available on a wide range of machines including Apollo, Cadmus, Harris, Masscomp, SUN and VAX (UNIX and VMS). Complete cross machine compatibility.

Franz Inc.
2920 Domingo, Suite 203
Berkeley, California 94705
(415) 540-1224

CIRCLE 55 ON READER SERVICE CARD

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

COMPILERS

AMSTERDAM COMPILER KIT

Subject: A complete Kit of compilers, cross compilers and assemblers.

The Amsterdam Compiler Kit is the only C and Pascal UNIX package which includes a wide range of native and cross tools. The Kit is also easily modifiable to support custom targets.

Features:

■ *C and Pascal compilers (native and cross) for UNIX machines.*
■ *Host and target machines include VAX™ 4.1/4.2 BSD, PDP™-11/V7, MC68000™ and 8086.™ Cross assemblers provided for 8080,™ Z80,™ Z8000,™ 8086,™ 6800,™ 6809,™ 68000,™ 6502 and PDP-11.*
■ *The Kit contains complete sources* of all programs, plus comprehensive internals documentation on how to make modifications needed to add a new program language or new target machine.*

**A source UNIX or C license is required from AT&T.*

Price:

Full Source System	*$9950*
Educational Institutions	*995*

Selected binaries are available – contact us with your machine type.

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 NJ). Telex: 709418. Japanese Distributor: SofTec 0480 (85) 6565. European Distributor: Modulator SA (031) 59 22 22

OEM terms available. Mastercard/Visa accepted.

UNIX is a trademark of AT&T Bell Laboratories. VAX & PDP 11 are trademarks of Digital Equipment Corp. MC68000, 6800 & 6809 are trademarks of Motorola Corp. 8080 & 8086 are trademarks of Intel Corp. Z80 & Z8000 are trademarks of Zilog, Inc.

CIRCLE 81 ON READER SERVICE CARD

UniPress Software
Your Leading Source for UNIX™ Software.

is not always as straightforward as one might like. The *preferred* statement is a message to the meta-level interpreter to prefer inferring on ugly before inferring on mean. This can be used to implement forms of searching, such as breadth-first searching.

Procedural attachment. Sometimes you will need access to more information or flexibility than is available inside MRS. You can access the underlying LISP system by using procedural attachment. The LISP system gives you access to the operating system calls and a faster language for computation. If the default structure of MRS makes a computation difficult or inefficient, you can attach a LISP procedure to do the work instead.

While procedural attachment gives you efficiency, it also removes MRS's ability to know what the procedure is doing. If you want MRS to gather information about what an attached procedure is doing, you must explicitly assert the facts you want MRS to know:

```
(assert '(totruep (time &x) get-time))
     ;totruep modifies how truep is
          done.
(defun get-time (x) (setq x (status
     ctime)))
(truep '(time )
```

This *truep* will return:

```
| Sun Apr 21 12:04:00 1985 |
```

The *ctime* function is an operating system call to the system clock. Reasoning systems can get access to the real world through this mechanism.

Because of all the things you might want to do with a predicate, several different mechanisms can attach a LISP procedure to a MRS predicate. But it is only important to know that this is possible. Let's see how it can be applied to a specific problem, which we'll call the frame problem.

Now that the basic programming tools available within MRS have been introduced, we can take a closer look at applying MRS to solve some more advanced problems. AI methods are usually search-intensive because they are employed to solve problems for which no direct algorithmic solution is available. The basic paradigm is to model problem solving as a controlled search through possible solutions.

Consider the design of a symbolic integration problem solver. The problem space can be defined as all mathematical expressions representable by the machine. Search is performed by the application of operators that transform mathematical expressions according to the rules of algebra and integration. The goal of the system is to locate in the space an expression that does not include an integral sign and that was formed by some sequence of operators applied to the initial problem expression.

The programmer is faced with two main tasks in design: first, that of knowledge representation, or how to describe the problem to the machine; and second, controlling the search so the problem is solved in a reasonable amount of time.

The construction of a problem solver involves representing two kinds of knowledge: state descriptions and operators. The two cannot be designated separately because the form of one determines the other. Basically, the states are described by predicates and the operators by rules. Search is performed by inference, either reasoning back from the goal state to the initial state or vice versa.

To clarify the important issues inherent in representation, a scheme for solving the standard AI toy problem of hobbits and orcs is presented. The problem is described as follows:

Two hobbits and two orcs find themselves on the north side of a river. They have agreed that they want to get to the other side. But the hobbits are not sure what else the orcs have agreed upon. So the hobbits want to manage the trip across the river in such a way that the number of hobbits on one side of the river is never less than the number of orcs on the same side. The only boat available holds just two people at a time. How can everyone get across the river without the hobbits getting eaten?

The similarity of this problem to symbolic integration can be seen by identifying the states and operators of this domain. A state describes some arrangement of the hobbits, orcs, and boat. The program searches the problem space containing all such arrangements. Rowing the boat over the river forms the operators that transform states.

We can imagine how a system would solve this problem. Beginning with the hobbits and orcs on the north side, various arrangements in the boat would be tried to form new states. At each new state a check would be made for termination (for example, all people on the south side) and

```
(assert '(if (ugly $x) (nasty $x ugly)))
(assert '(if (mean $x) (nasty $x mean)))
(assert '(template (nasty &x &y)(&x is nasty because he is &y)
(setq preferred t)
(assert '(preferred (bcdisp ((ugly &x)).1)
               (bcdisp ((mean &y)). &m)))
(find '(nasty $x $y))

ragashak is nasty because he is ugly
```

Listing 2.

```
    (goal-state  {statedescription})
    (illegal-state {statedescription})
    (possible-operator operator-n {statedescription} {newstatedescription} )
    (if (or (goal-state {statedescription})
          (and (unprovable (illegal-state {statedescription}))
               (possible-operator $operator {statedes} {newstatedes} )
               (hobbits-orcs $ops {newstatedes})))
        (hobbits-orcs ($operator . $ops) {statedescription} ))
```

Listing 3.

illegal states (hobbits get eaten). If the search is to continue, new operators are applied and the process repeated recursively. We can develop a rule scheme to perform this search, as shown in Listing 3.

Backward chaining inference can be used to solve this problem. The user types:

```
(truep '(hobbits-orcs $operators
    {initialstate}))
```

The system will return a binding list with *$operators* bound to a list of suitable operators.

A state describes the location of all the hobbits, orcs, and the boat in the domain. Other information, such as the count of hobbits and orcs on each side, may be included in the description to simplify the rules. One way to represent a state is to explicitly mention all the individual descriptive facts in the rules and predicates:

```
(hobbits-orcs $operators $sideboat
    $sideorc1
$sidehobbit1 $northhobbitcount...)
```

and initially call the problem solver with:

```
(hobbits-orcs $operators north north
    north 2 ......................)
```

The method looks a bit clumsy with this problem, but it is quite suitable for very simple domains. As domains increase in complexity and state descriptions become more involved, this approach becomes impractical.

This method has two main disadvantages. First, the memory requirements become excessive during the search because of the many copies of the full state description stored on the agenda. Second, solution time increases because of the time spent copying unchanged information from one state description to another.

The difficulty with this approach is apparent in real world domains such as robots whose states are described by thousands of facts. This problem—the frame problem—is well known in AI and occurs in all nontrivial domains when a system has to reason about changes.

An elegant solution to this problem was proposed in 1970. The basic idea of the strips method (named after the robot planning system) is to represent all facts describing states as individual predicates in a global data base and to use lookups to access data and *assert* to make changes.

In this rule scheme, the state descriptions are replaced by state variables and

Logic programming: the art of reasoning

Before asking the question "What is logic programming?" we first must ask "What is logic?"

In terms of programming, logic can be thought of as the process of reasoning, both rightly or wrongly, toward a conclusion. To bring this definition into focus, let's look at logic in terms of the kinds of problems it can solve.

Syllogisms are types of logical arguments having two premises and a conclusion. For example, consider the following:

All men are tall.
All tall people wear hats.

All men wear hats.

The first two lines are the premise, the last the conclusion. The basic question here is if the premise is true, will the conclusion be true? Or, in more precise terms, does the conclusion logically follow from the premise?

Logic can prove a conclusion based on a certain starting premise by showing that the conclusion logically follows the premise. As in mathematics, many of the more advanced theories of logic are derived in this way from base-level postulates.

Logic, as applied to natural language interpretation, is very useful for understanding language. This understanding is essential for certain forms of processing, such as language translation, and for deriving meaning from the written and spoken word.

These examples of the types of problems logic addresses show that it is a very powerful tool. To better understand how the science of logic has progressed to where it is now, let's look at where it started.

History shows Aristotle to be the father of logic. In fact, much of today's study of logic still consists of rules and assumptions made by Aristotle and some of his followers. Between Aristotle's time and the 1800s, his work stood without serious challenge. This changed with the work of two great minds. Augustus De Morgan, born in 1806, was a mathematician and philosopher who delved into many aspects of logic and mathematics. He also was instrumental in bringing to attention the work of a hitherto unknown school teacher, George Boole. Born in the late 1700s, Boole's work concentrated on the algebraic properties of logic.

Many of the methods and theories put forth by Boole and De Morgan stand unchanged to this day. Their fields of expertise included mathematics, logic, and many forms of engineering analysis, such as electrical circuit analysis.

Now, you may ask, what does logic programming have to do with logic? The study of logic may be interesting, but why apply it to programming? Why take the trouble to change? Programming in logic does not necessarily fit into the traditional von Neumann computer model—that is, that one instruction follows another.

The answer is that some problems can be described very easily using the constructs of logic. In logic programming, the process of specifying how the solution should look is almost equivalent to writing the program.

Logic programming brings to the computer power to attack problems that previously were difficult, if not impossible, to understand. From a programmer's standpoint, logic can be thought of as a system of rules and facts.

Rules and facts take the place of the traditional statement used by other programming languages. Facts are simply statements of constants (for example, "Lee is tall"). Rules are traditional logical statements (for example "A man is tall if he is thin"). In addition, rules can be interpreted in two ways: procedurally, as in the von Neuman tradition, and semantically. For example, the rule "A man is tall if he is thin" can be interpeted procedurally as "To find a thin man, first find a tall one" or semantically as "All thin men are tall men."

In addition to their the ability to use logic, some logic programming languages allow control over how the inference process is executed. These languages are sometimes referred to as meta-logical, because they can control the flow of logic.
For more information on logic programming, a good general reference is W.F. Clocksin and C.S. Mellish's *Programming in Logic,* available from Springer-Verlag.

By William Lee Duncan

William Lee Duncan is an independent UNIX consultant and has just finished a year of graduate school at Oregon State Univ. He received a B.S. in electrical engineering from Oregon State in 1981 and was employed with Hewlett-Packard for two years.

the rules extended to look up the relevant values. The initial state and termination recognition rule presented in the example problem can be seen in Listing 4.

Changes to the state are achieved by asserting new facts into the data base with a new state variable and altering the way facts are looked up. Consider the effect of rowing the boat to the south side with both hobbits aboard. The possible-operator rule will generate a new state variable and assert the following facts:

```
(onside state-1 south hobbits Bilbo)
(onside state-1 south numberofhobbits
     2)

(nextstate state-0 state-1)
```

To look up the state of the world at state-1, the system performs a search—for example, for the statement *(onside state-1 south countofhobbits$count)*, the value 2 will be returned directly. But if you want to look up *(onside state-1 north numberoforcs $count)*, the system will first check state-1 and, on finding nil, will use the *nextstate* relationship to find the previous state and lookup in state-0. Generally, the search will continue until a value is found.

MRS has many advantages over languages such as PROLOG for implementing systems like this. The lookup routines can be implemented as procedural attachments as shown in Listing 5. The method in Listing 5 allows MRS programs to solve problems in large, complex domains where the conventional approach is infeasible.

Controlling the search through the problem space often is not just preferable to allow speedup but can be essential for termination. In the example problem, for instance, without the ability to control the search, the system might have tried to row the boat back and forth infinitely with one hobbit aboard.

A spectrum of methods is available for controlling the search. Conventional algorithmic solutions have built-in controls and usually solve problems in ideal times. Many problems, although not algorithmicly solvable, allow heuristic knowledge that can be applied to select one operator or state over another. Where no additional knowledge is available, systems revert to pure generate and test, with solution times greatly increasing.

One of the major achievements of AI research over the last 25 years has been the realization of the need for additional knowledge to guide searches through problem spaces. Speedups in the order of magnitudes can be gained by the addition of some simple heuristic knowledge. In the hobbits and orc problem, tremendous speedups can be achieved by making the system prefer the operator that carries two people south and only one north over any other operator.

Because of the importance of search control, it is surprising that MRS is the only logic programming language to directly incorporate it. PROLOG solves all its problems using a depth-first generate and test, with the static ordering of clauses providing the only direct control.

MRS allows the user to impose an ordering over the tasks on the agenda. To order search by depth, the main problem solving clause in the example problem (such as *(hobbits-orcs . . .)*) needs an additional slot to hold the depth information, which is incremented at each recursive call.

Preferred rules can be written to select the lowest depth to give breadth-first search or to select the greatest value to give depth-first search. The numerical value in the clause can be computed by an evaluation function that refers to the current state giving best-first search.

If you would like a copy of MRS, it is available for $500 ($200 for universities). For more information, contact:

Margaret Timothy
Symbolics Systems Resources Group
Stanford University Medical Center
Room TB 105
Stanford, Calif. 94025 ■

John Sechrest has a B.S. in math and computer science from the Univ. of Illinois at Champaign-Urbana. Formerly employed with Hewlett-Packard, he is currently a lab coordinator with Oregon State Univ. in Corvallis, Ore.

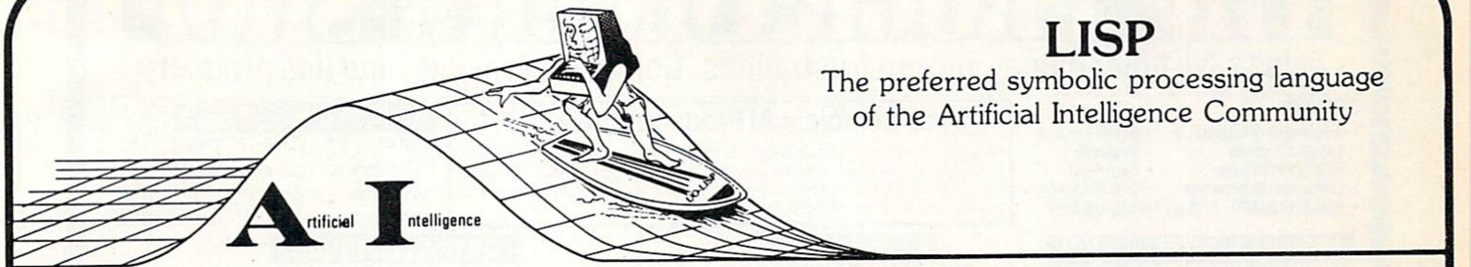
Nick Flann has an electric and electronic engineering degree from Coventry Polytechnic in England and is currently working on a masters in artificial intelligence at Oregon State Univ.

```
(onside state-0 north orcs ragashak)
(onside state-0 north numberoforcs 2)
(onside state-0 south numberoforcs 0)
(onside state-0 north hobbits Bilbo)....

   (if (and (onside $state south numberoforcs 2)
            (onside $state south numberofhobbits 2))
       (goal $state))

   (if (and (onside $state $anyside numberoforcs $norcs)
            (onside $state $anyside numberofhobbits $nhobbits)
            (> $norcs $nhobbits))
            (illegal-state $state))
```

Listing 4.

```
(tolookup (onside &state &side &type &value) strips)

(defun strips
   (state side type value)
   (or (lookup `(onside-g ,state ,side ,type ,value))
       (strips   (lookupbdg '$nextstate `($nextstate  $nextstate ,state))
             side type value)))
```

Listing 5.



LISP

The preferred symbolic processing language of the Artificial Intelligence Community

catch the next micro-wave with

UO-LISP

Not "just another pretty dialect" but the most powerful implementation of LISP available in the micro market place. For the professional engineers, researchers, and educators, UO-LISP maintains the power and flexibility inherent in LISP while providing the expected functionality of mainframe LISP systems. (+) **UO-LISP steps beyond the competition and provides a real source to native code compiler.**

CPU Family	Operating System	Production System	Learn System
8086	MS-DOS	150^{00}	85^{00}
	PC-DOS	150^{00}	85^{00}
	CPM/86	*available soon*	
Z80	CPM	125^{00}	85^{00}
	TRS DOS	80^{00}	N/A

For MORE DETAIL AND TO ORDER:
Send for *FREE brochures and order forms.*

NORTHWEST COMPUTER ALGORITHMS
P.O. Box 90995, Long Beach, California 90809
(213) 426-1893

CIRCLE 39 ON READER SERVICE CARD

™ + = AI

ExperTelligence™ has the "tools" to transform your Macintosh™ into a powerful Artificial Intelligence workstation. **ExperLisp**™ is the first complete implementation of LISP on a microcomputer. Developed on a Symbolics 3600,™ the compiler generates efficient MC68000 code providing speed and function ideal for the development and delivery of sophisticated AI applications. **ExperOPS5**,™ by Science Applications International Corporation, is a complete implementation of the well-known OPS5 expert systems building tool. It provides a fast and efficient method for constructing complex Expert Systems.

ExperLogo™ features 3-D and spherical graphics, English-like commands and shares the speed and function of ExperLisp. In the classroom or in the lab, ExperLogo provides an environment for discovery and exploration for children and developers alike.

Call today for more information about these and other innovative AI products.

™

ExperTelligence, Inc.
559 San Ysidro Road
Santa Barbara, CA 93108
Tel: 805/969-7871.

Macintosh is a trademark licensed to Apple Computer, Inc.
Symbolics 3600 is a registered trademark of Symbolics, Inc.

CIRCLE 30 ON READER SERVICE CARD

THE PROGRAMMER'S SHOP™

helps save time, money and cut frustrations. Compare, evaluate, and find products.

SERVICES
- Programmer's Referral List
- Compare Products
- Help find a Publisher
- Evaluation Literature free
- BULLETIN BOARD - 7 PM to 7 AM 617-826-4086
- Dealer's Inquire
- Newsletter
- Rush Order
- Over 700 products

Free Complete AI Products Literature

Evaluate products Compare competitors. Learn about new alternatives. One *free call* brings information on just about any programming need. Ask for any LISP, PROLOG, SMALLTALK, EXPERT SYSTEMS. Includes products for learning and for sophisticated development. Macintosh and MSDOS. 20 + pages.

RECENT DISCOVERIES

SMALL TALK for PCDOS - "Methods" has objects, windows, browser, inspector. PCDOS $239

ARTIFICIAL INTELLIGENCE

ExpertLISP - Interpreter: Common LISP syntax, lexical scoping, toolbox, graphics. Native code COMPILER. 512K MAC $465

ExpertEASE - Expert system tool. Develop by describing examples of how you decide. PCDOS $625

EXSYS - Expert System building tool. Full RAM, Probability, Why, serious, files PCDOS $275

GC LISP - "COMMON LISP", Help, tutorial, co-routines, compiled functions, thorough. PCDOS Call

INSIGHT 1 - Expert Sys. Dev't, decent PCDOS $ 95

M Prolog - full, rich, separate work spaces. MSDOS $725

PROLOG-86 - Learn fast, Standard, tutorials, samples of Natural Language, Exp. Sys. MSDOS $125

TLC LISP - "LISP-machine"-like, all RAM, classes, turtle graphics 8087. CP/M-86, MSDOS $235

EDITORS FOR PROGRAMMING

BRIEF Programmer's Editor - undo, windows, reconfigurable, macro programs, powerful. PCDOS $195

VEDIT - well liked, macros, buffers, CPM-80-86, MSDOS, PCDOS $119

MACINTOSH

We evaluate, carry every available programmers product. Ask for a packet describing OVER 20 PRODUCTS

C LANGUAGE

C Terp Interpreter by Gimbel, full K&R, .OBJ and ASM interface, 8087. MSDOS $275

INSTANT C - Interactive development - Edit, Source Debug, run. Edit to Run - 3 Secs. MSDOS $445

"INTRODUCING C" - Interactive C to learn fast. 500 page tutorial, examples, graphics. PCDOS $ 95

MEGAMAX C - native Macintosh has fast compile, tight code, K&R, toolkit, .OBJ, DisASM MAC $275

Wizard C - Lattice C compatible, full sys. III syntax, lint included, fast, lib. source. MSDOS $450

C ADDONS

APPLICATION TOOLKIT by Shaw - Complete: ISAM, Screen, Overlay mgnt, report gen, Strings, String math. Source. CPM, MSDOS $475

COMMUNICATIONS by Greenleaf ($159) or Software horizons ($139) includes Modem7, interrupts, etc. Source. Ask for Greenleaf demo.

C SHARP Realtime Toolkit - well supported, thorough, portable, objects, state sys. Source MANY $600

C Index + - full B+Tree, variable length fields. Source, no royalties. MSDOS $369

PC/LINT - Small, big model. Batch option. Lattice, C86 MSDOS $ 95

FORTRAN LANGUAGE

MacFORTRAN - full '77, '66 option, toolbox, debugger, 128K or 512K, ASM-out option MAC $375

RM/Fortran - Full '77, BIG ARRAYS, 8087, optimize, back trace, debug. MSDOS $525

Ask about Microsoft, Supersoft, others.

OTHER LANGUAGES

ASSEMBLER - ask about FASM-86 ($95), ED/ASM ($95) - both are fast, compatible, or MASM ($125), improvements.

BetterBASIC all RAM, modules, structure. BASICA-like PCDOS $185

SNOBOL 4 + - great for strings, patterns. CPM86, MSDOS $ 85

SUPPORT PRODUCTS

BASIC DEVELOPMENT SYSTEM - (BDS) for BASICA; Adds Renum, crossref, compress. PCDOS $115

CODESIFTER - Execution PROFILER. Spot bottlenecks. Symbolic, automatic. PCDOS $109

FASTER C - Lattice users eliminate Link step. Normal 27 seconds. Faster C in 13 sec. MSDOS $ 95

PLINK-86 for Overlays, most lang., segment control. MSDOS $325

PS MAKE by Unipress - Interactive or batch. Full MAKE. MSDOS $129

"C" LANGUAGE

	OUR PRICE
MSDOS C86-8087, reliable	call
Instant C - Inter., fast, full	445
Lattice C - the standard	call
Microsoft C 3.0 - new	279
Williams - debugger, fast	call
Wizard C - full, fast	450
CPM80 - EcoPlus C - faster, SLR	275
BDS C - solid value	125
MACINTOSH: Hippo II	375
Megamax - optimizer, full	275
Consulair's MAC C, toolkit	395

Compare, evaluate, consider other Cs

BASIC

	RUNS ON	
BASCOM-86 - MicroSoft	8086	279
CB-86 - DRI	CPM86	419
Data Manager-full source	MSDOS	325
CADSAM-Full Btree, source	MSDOS	150
InfoREPORTER-multifile	PCDOS	115
Prof. Basic-Inter., debug	PCDOS	89
SCREEN SCULPTOR	PCDOS	115
TRUE BASIC - ANSI	PCDOS	125

Ask about ISAM, other addons for BASIC

SERVICE

ALL PRODUCTS - We carry 700 products for MSDOS, CP/M 86, CP/M 80. Macintosh and key products for other micros.

EDITORS Programming

	RUNS ON	OUR PRICE
BRIEF - Intuitive, flexible	PCDOS	195
C Screen with source	86/80	75
Epsilon - like EMACS	PCDOS	195
FINAL WORD-for manuals	86/80	215
PMATE - powerful	8086	185
VEDIT - full, liked	86/80	119
XTC - multitasking	PCDOS	95

COBOL

Dig. Res-decent	MSDOS	525
Macintosh COBOL - Full	MAC	1850
MBP-Lev II, native, screen	MSDOS	885
Micro Focus Prof. - Full	PCDOS	call
Microsoft - Lev II, no royal	MSDOS	500
Ryan McFarland - portable	MSDOS	695

Ask about program generators.

LANGUAGE LIBRARIES

GRAPHICS: Halo for Turbo Postal	MSDOS	95
GRAPHMATIC - 3D, FTN, PAS	PCDOS	125
MultiHALO - fast, full-all lang.	PCDOS	220
File MGNT: BTrieve-all lang.	MSDOS	215
CIndex + - source, no royal	86/80	369
CTree - source, no royal	ALL	369
dBC ISAM by Lattice	8086	229
dB VISTA - "Network" Structure	MSDOS	465
PHACT - up under UNIX, addons	MSDOS	225
OTHER: C Utilities by Essential	MSDOS	129
Greenleaf - 200 +	MSDOS	159
SOFT Horizons - Blocks I	PCDOS	139
SCREEN: CURSES by Lattice	PCDOS	125
MetaWINDOW - icons, clip	PCDOS	139
PANEL - many lang. term	MSDOS	249
ProScreen - windows, source	PCDOS	415
Turbo V - Greenleaf C. fast	PCDOS	159
Windows for C	MSDOS	175

FORTRAN

	RUNS ON	OUR PRICE
MS FORTRAN-86 - Impr.	MSDOS	239
DR Fortran-86 - full '77	8086	249
PolyFORTRAN-XREF, Xtract	PCDOS	165

OTHER PRODUCTS

Advanced Trace 86 - Symbolic	PCDOS	149
Assembler & Tools - DRI	8086	159
Atron Debugger for Lattice	PCDOS	395
C Helper: DIFF, xref, more	86/80	135
CODESMITH-86 - debug	PCDOS	129
MacASM - full, fast, tools	MAC	115
MBP Cobol-86 - fast	8086	885
MicroProlog - improved	MSDOS	185
Micro: SubMATH - FORTRAN full	86/80	250
Microsoft MASM-86	MSDOS	125
Multilink - Miltitasking	PCDOS	265
PC FORTH - well liked	MSDOS	219
Pfinish - Profile by routine	MSDOS	345
PFIX-86 Debugger	MSDOS	169
PL1-86	8086	495
Polylibrarian - thorough	MSDOS	95
PolyMAKE	PCDOS	95
TRACE86 debugger ASM	MSDOS	115
ZAP Communications-VT100, TEK 4010, full xfer	PCDOS	65

Note: All prices subject to change without notice. Mention this ad. Some prices are specials. Ask about COD and **POs**. All formats available. UNIX is a trademark of Bell Labs.

Call for a catalog, literature, and *solid value*

800-421-8006

THE PROGRAMMER'S SHOP™

128-I Rockland Street, Hanover, MA 02339

Visa Mass: 800-442-8070 or 617-826-7531 MasterCard 8517

CIRCLE 84 ON READER SERVICE CARD

⊗○⊗○ PRODUCT BINGO ⊗○⊗○

By Doug Millison

Each month Product Bingo features the latest in new software and hardware products of interest to COMPUTER LANGUAGE *readers. Send new product information to Doug Millison, Product Bingo,* COMPUTER LANGUAGE, *131 Townsend St., San Francisco, Calif. 94107.*

Thoroughbred BASIC for IBM PC/AT

Thoroughbred BASIC, a business basic interpreter designed for 16-bit, multiuser microcomputers, is now available from **SMC Software Systems** for the IBM PC/AT running Microsoft's XENIX 3.0 operating system.

SMC Software Systems, P.O. Box 0600, Basking Ridge, N.J. 07920, (201) 647-7000.

CIRCLE 101 ON READER SERVICE CARD

Get Smart/C

Smart/C incorporates AI techniques in a fully integrated precompilation development environment for C. Smart/C prices range from $500 for the IBM PC to $10,000 for VAX 11/780.

AGS Computers Inc., 1139 Spruce Dr., Mountainside, N.J. 07092, (201) 654-4321.

CIRCLE 102 ON READER SERVICE CARD

Multiwindow, multiuser C

Interactive C is a multiwindow, multiuser, full-featured C language development system for the IBM PC, priced at $395.

IMPACC Associates, P.O. Box 93, Gwynedd Valley, Pa. 19437.

CIRCLE 103 ON READER SERVICE CARD

True source debugger for C

Programmers using the Lattice C compiler will be interested in **C Debugger**, priced at $165 plus shipping and handling.

Micro-Software Developers Inc., 214½ W. Main St., St. Charles, Ill. 60174, (312) 377-5151.

CIRCLE 104 ON READER SERVICE CARD

UNIX-compatible Forth

u4th brings a portable, standard Forth to the UNIX and XENIX world.

Ubiquitous Systems Inc., 13333 Bel-Red Rd. N.E., Bellevue, Wash. 98005, (206) 641-8030.

CIRCLE 105 ON READER SERVICE CARD

MacFORTRAN for guess what?

MacFORTRAN, an ANSI FORTRAN 77 compiler with debugger, was designed for the Macintosh and features full access to the Mac toolbox.

Absoft Corp., 4268 N. Woodward Ave., Royal Oak, Minn. 48072, (313) 549-7111.

CIRCLE 106 ON READER SERVICE CARD

B-tree indexing

B-Tree Library is available with full documentation, full C source, and phone support in most formats for $75.

Softfocus, 1277 Pallatine Dr., Oakville, Ont., Canada L6H 1Z1, (416) 844-2610.

CIRCLE 108 ON READER SERVICE CARD

Utilities power Turbo Pascal

Turbo Pascal users can power-up with **TurboPower Utilities**, priced at $95 for the IBM PC family and compatibles.

TurboPower Software, 478 W. Hamilton Ave., Ste. 196, Campbell, Calif. 95008, (408) 378-3672.

CIRCLE 109 ON READER SERVICE CARD

Modula-2 for $80.88

$80.88 gets you a full-featured Modula-2 programming environment for the IBM PC family and compatibles.

Interface Technologies Corp., 3336 Richmond Ave., Ste. 200, Houston, Texas 77098, (713) 523-8422.

CIRCLE 110 ON READER SERVICE CARD

CTOS/BTOS version of CLOUT

Clout, a natural language data base, is now available for CTOS/BTOS-based microcomputers.

Microrim Inc., 3380 146th Pl. S.E., Bellevue, Wash. 98007, (206) 641-6619.

CIRCLE 111 ON READER SERVICE CARD

C program analyzer for MS-DOS

Pre-C, a new MS-DOS/PC-DOS program analyzer for C users, is priced at $395.

Phoenix Computer Products Corp., 1416 Providence Highway, Ste. 220, Norwood, Mass. 02062, (617) 762-5030.

CIRCLE 112 ON READER SERVICE CARD

HCR/PASCAL, WIRTH ITS WEIGHT IN C

PASCAL Originally designed by Niklaus Wirth, is now available for a wide range of UNIX™ processors. HCR/PASCAL conforms closely to industry standards, passes all conformance tests in the PASCAL Validation Suite. Supports multiple module programs, a dynamic string package, and direct random file access.

C is the standard language of UNIX, HCR/PASCAL is written in C and translates PASCAL into C producing efficient optimized code. This approach allows direct interaction with the UNIX environment and offers a high degree of portability.

UNIX is a powerful yet flexible operating system environment. HCR/PASCAL is available today on a diverse range of UNIX hardware: AT&T 3B™ series, the NCR Tower,™ DEC PDP-11/VAX,™ and others. HCR has a growing line of UNIX software including business applications. We back up all our software with full support. To find out how we can put HCR/PASCAL, C, and UNIX together for you, call or write:

Human
Computing
Resources
Corporation

HCR

10 St. Mary Street, Toronto, Ontario, Canada M4Y 1P9 (416) 922-1937

UNIX is a trademark of Bell Laboratories. PDP-11, VAX, and DEC are trademarks of Digital Equipment Corporation. AT&T 3B is a trademark of American Telegraph & Telephone. NCR Tower is a trademark of NCR Corporation.

CIRCLE 93 ON READER SERVICE CARD

Knowledge Systems for the IBM PC, Part I

By Ernie Tello

Expert systems technology is one of the fastest moving and most widely discussed areas of computer software. Strictly speaking, an expert system is a finished knowledge system application that has been perfected until it achieves expert-level performance. However, the term is often used in a loose way to mean the tools designed specifically for developing these applications as well as for any and every application that uses this same type of technology.

Numerous legitimate applications of this technology could not be called expert systems in the true sense by any stretch of the imagination. So it seems much less confusing to consider expert systems an important subclass of a technology that could be called knowledge systems or knowledge-based software.

This review will attempt to give you an overview of this software system as well as offer a survey of the first major wave of knowledge system development tools for PCs. This month, in Part I, we will look at the following categories of expert systems: decision modeling software, language extension packages, example-driven decision tree systems, and small production system tools.

Next month, in Part II, we will explore rule-oriented mathematical modeling systems, intermediate-level systems, and advanced systems. The following products will be reviewed in Part II: Insight-2 from Level Five Research Inc., Expert Systems International's ES/P-Advisor, the TIMM PC from General Research Corp., Lotus/Software Arts' TK!Solver, REVEAL from McDonnell-Douglas, KES from Knowledge Engineering System, and Teknowledge Inc.'s M.I.

In terms of knowledge, there are two basic types of software applications: those that incorporate knowledge to help a user better handle problems and those that are primarily efficiency tools to which a user must apply his or her own knowledge and skill, such as word processors, spreadsheets, and programming languages. A superb word processor can take much of the drudgery out of writing, but a poor writer will still write just as poorly with it. With knowledge-based software, the idea is that the knowledge built into it can and should help the user to perform better.

Although expert systems technology is a by-product of artificial intelligence research, not all knowledge-based software is. Often when a program is described as AI, a user or programmer who tries it will expect some awe-inspiring omniscience to emanate from its every movement. But even the best expert systems developed so far are only expert in their ability to apply expert knowledge to solving certain types of problems. They are in no sense a replacement for living, breathing experts who can talk to you and think about your problems. Let's take a brief look at some of the major types of knowledge-based systems to see what they are and what they are capable of doing.

Production systems. Most of the software available for developing expert systems is aimed at creating some type of rule-based production system. Usually such a system will have at least three components: a rule base, an inference engine, and a consultation shell. The consultation shell is the end-user environment for the finished application.

In a given session, the system will pose certain questions to the user. Depending upon the answers given, the system is directed to a particular outcome where a result is displayed. The result may be a choice of several alternatives, a diagnosis of the cause of trouble, a prediction that various events have a certain probability of occurring, or something along these lines.

One of the things that production-rule systems offer that hard-coded programs generally do not is the ability to justify their answers and results. This is extremely important, because without it a user would be expected either to accept the results on faith or ignore them. The more a user is able to explore the way a system arrived at a particular result, the more the user is able to learn about the problem and the better the user is able to make a decision and analyze the results.

But as important as a justification facility is, even more important is the amount of detail and flexibility a system provides in the final outcome. The whole point is to help solve a problem in as expert a manner as possible. It is the power of the representation language used to provide the rule base that determines the quality of the outcome.

Some important initial questions should be asked about the representation language:

- Are there symbolic variables that allow the general statement of rules for whole ranges of lookup values?
- Does it allow the calculation of numerical values on the basis of a user's input?
- Does it provide for the entry of facts as well as rules?
- Does the language provide for *OR* operators between the antecedent conditions of a rule?

It seldom takes an expert to tell you what you need for your particular situation and requirements. But to tell you how much you need requires expertise. Yet if the representation language has no math capability, there is no way such expertise can be included in the system. If the system can only handle rules about constant numbers and not variables, the number of rules needed may get astronomical. Similarly, the presence of *OR* operators for conditions often can keep the number of rules down. Otherwise, a new rule would have to be entered for each of the alternate conditions. The fewer rules a production system has to search through, the faster it can run. So the number of rules an expert system tool is capable of handling actually means very little. What is more important is how well and how easily the tool can be used to create powerful and compact rules with the system.

Closely tied to a given representation language is the inference engine, which is the control center for whatever degree of intelligence the production system can exhibit. The two main types of inference procedures are forward chaining and backward chaining. Although it would be nice to have a system capable of both, so far there are no knowledge engineering tools for microcomputers that offer both as complementary tools to design single knowledge systems. At best, there are high-level languages that allow develop-

BRING BACK ERASED FILES

FILE SAVIOR ™

is a disk utility you can't afford to be without!

- Makes recovering an erased file easy... as long as you haven't written a new file on top of it
- Display and edit any disk sector – in hex, Ascii or Ebcdic
- Search a file or your whole disk for a text string
- IBM PC/XT/AT and true compatibles
- DOS 1.0 thru 3.1
- Any DOS-formatted diskette or hard disk – up to 33 meg

49⁹⁵ VA residents add 4% sales tax

Technisoft

1710 Allied Street, Suite 37
Charlottesville, VA 22901
(804) 979-6464

FAST SCREEN OUTPUT FOR TURBO PASCAL

FASTSCREEN ™

is a set of inline assembler and Pascal procedures for Turbo Pascal users.

- Display an entire screen or window almost instantly
- Process multi-field input screens that give your user full cursor control
- Sample program uses Conway's LIFE to illustrate use of procedures
- All source code included
- Color and monochrome support
- IBM PC/XT/AT and true compatibles
- DOS operating system

29⁹⁵ VA residents add 4% sales tax

Technisoft

1710 Allied Street, Suite 37
Charlottesville, VA 22901
(804) 979-6464

Turbo Pascal is a registered trademark of Borland International

CIRCLE 83 ON READER SERVICE CARD

ers who know what they're doing to implement both. Let's take a look at what these two types of procedures involve.

The main requirements for an inference engine are:

■ A way of determining when and where to start

■ A procedure for searching toward a result

■ A way of determining that a result has been found or that no result can be found.

In a backward chaining inference engine, the reasoning process begins with a conclusion, or *Then* part of a rule, which is designated as a goal. A very simple system might just pick the conclusion of rule 1 as the first goal and go on down the list. In a more sophisticated system, the knowledge engineer designates what the goal is to be, and the inference engine looks for the rules that have that goal as their conclusion.

Once the goal has been determined, the inference procedure attempts to prove true those antecedent conditions that establish that goal. Often the process will stop once one rule has been found where all the antecedents that prove the goal are true. More sophisticated systems allow attributes that permit multiple values. Sometimes you might want all the possible solutions to a goal—for example, if the goal were a way to get $10,000 in one day. In this case, the system would attempt to find as many values as it could for the goal.

Forward chaining inference engines approach the problem in the opposite manner. They are bottom-up, or data-driven, systems. They begin by trying to find antecedents or conditions in the *if* part of rules that are true facts. In this case, it is not sufficient to have just pointers to the goal and the current rule; it is necessary to have an inventory of known facts that is maintained in a list known as the working memory. The antecedents of rules are compared with the contents of the working memory to determine if they are true. When all the antecedents of a rule compare favorably, the conclusions of the rule are added to the contents of the working memory and the process continues.

Decision trees. Two types of software packages discussed here directly incorporate decision trees: decision modeling programs and example-driven, rule induction programs. Although production-rule systems usually result less directly in data structures that are similar to decision trees, the flexibility of their use and the explicit nature of the knowledge used to build them are their key advantages. In effect, decision-tree software takes a shortcut. If a tree structure is to result anyway, the approach of this type of software is to generate the structures as easily and efficiently as possible, with the least amount of trouble to both the developer and user. But in doing this, decision-tree software gives up the ability to handle as broad a range of problems and to give transparent reasons for the results.

Decision modeling programs are significantly different from most of the products discussed here. These programs belong to the category of decision support software. Although this is clearly an important class of knowledge-based software and uses decision trees, in most other respects these programs are quite different from expert system development tools.

An example-driven, rule induction system is one that can take the logical skeleton of a problem—in terms of the value certain variables take as other variables change—and build a decision tree with the ability to induce a general rule about how one variable varies with another based on specific examples. In principle, the advantages of such a system are that it is rather straightforward to use, fast, and efficient. It is important, though, to carefully evaluate particular implementations of this product to ensure that these advantages do in fact exist.

Data base interfaces. To get the most out of a knowledge system, the system should be capable of interfacing with files of a powerful data base. Most businesses do not want to enter their data all over again, and it is not productive for users to have to continually answer questions about information that is already on the system.

At the minimum, an interface capability should execute external programs and return values to the running knowledge system application. This allows the external program to query the user or a data base or read hardware sensors for values, perform calculations on them, and then return a value to the rule that requested it. But a fully configured interface capability would allow the expert system itself to send a fairly extensive list of parameters to an external environment for activating nested batch files and performing sophisticated multiple queries to a relational data base.

In terms of real-time use, it is often not adequate for the expert system to talk to the outside world only when it needs information. Alarms that come in have to be evaluated by the rules, and there has to be a way for the system to handle a rapid foray of such alarms without jumping from one to the other and never reaching any conclusions.

What makes knowledge systems distinctive, then, is not just their ability to incorporate sophisticated knowledge. They are also distinctive because of their capacity to do it explicitly and in such a way that it is relatively easy to add and subtract knowledge from them and to partition them so that certain sections can accept alternate knowledge blocks. In this

A FULL C COMPILER FOR $49⁹⁵

The Ecosoft Eco-C88 compiler for the 8088 and MSDOS is going to set a new standard for price and performance. Consider the evidence:

Compiler	Eco-C88	Latice (1)	C86 (1)
Seive	13	11	13
Fib	44	58	46
Deref	13	13	–
Matrix	21	29	27
Price	$49.95	$500.00	$395.00

(1) **Computer Language,** Feb., 1985, pp.73-102. Reprinted by permission.

The Eco-C88 compiler is a full K&R C compiler that supports all data types and operators (except bit fields). Now look at the other features we offer:

★ 8087 co-processor support using a single library. If you install an 8087 later, the software will use it without having to recompile.
★ A robust standard library with over 150 functions, including transcendentals, color, and others.
★ OBJ output for linking with the MSDOS linker (LINK).
★ Error messages in English – no cryptic numbers to look up. A real plus especially if you're just getting started with C.
★ Easy-to-read and complete user's manual.
★ Works with all IBM and compatibles running MSDOS 2.0 (or later).
★ Plus many other features.

For $10.00 more, we will include the source code for the C library functions (excluding transcendentals). For an additional $15.00, we will include our ISAM file handler in OBJ format (as published in the **C Programmer's Library,** Que Publishing). The discount prices for the library source and ISAM only apply at the time the compiler is purchased. Please add $4.00 to cover postage and handling. To order, call or write:

ECOSOFT INC.

Ecosoft Inc.
6413 N. College Avenue
Indianapolis, IN 46220
(317) 255-6476

MasterCard VISA AMERICAN EXPRESS

Eco-C (Ecosoft), MSDOS (Microsoft), UNIX (Bell Labs), CP/M (Digital Research), Z80 (Zilog), 8086, 8087, 8088 (Intel).

CIRCLE 17 ON READER SERVICE CARD

Pascal Users—Save Time & Errors
(with PascalPac™ and tidy™)

For the IBM PC, XT, AT.

A major step toward paperless programming!
Debug on the screen!
Neater programs!

PascalPac™
X-REF creates cross reference table.
X-RAY browses cross reference and program simultaneously.
X-PRNT is a versatile listing program.
X-PEEK browses programs or text files.
One version of PascalPac supports Microsoft and TURBO PASCAL.

tidy™
Program formatter makes a PASCAL program easier to read, understand and modify. **In use for over one year in major companies and programming organizations.** Available in separate versions for Microsoft and TURBO PASCAL.

"tidy is a lightning-fast Pascal formatter from Major Software."
"Of the products we've seen, we felt that tidy is the one most programmers will prefer."
PC Tech Journal

PascalPac....**$69;** Separate Modules...**$60;** tidy-Turbo...**$49;** tidy Microsoft...**$69;** Shipping...**$5**

To order: VISA/MasterCard orders. **Call (415) 941-1924.** Or mail check/money order to:
Major Software, 66 Sylvian Way, Los Altos, CA 94022

CIRCLE 14 ON READER SERVICE CARD

BORED?...

...waiting for C programs to compile and link?

Use **C-terp**
the **complete** C interpreter

This is the product you've been waiting (and waiting) for!

Increase your productivity and avoid agonizing waits. Get instant feedback of your C programs for debugging and rapid prototyping. Then use your compiler for what it does best...compiling efficient code ...slowly.

C-terp Features

- Full K&R C (no compromises)
- Complete built-in screen editor-- no half-way house, this editor has everything you need such as multi-files, inter-file move and copy, global searching, auto-indent, tab control, and much more.
- Fast--Linking and semi-compilation are breath-takingly fast. (From edit to run completion in a fraction of a second for small programs.)
- Convenient--Compiling and running are only a key-stroke or two away. Errors direct you back to the editor with the cursor set to the trouble spot.
- Object Module Support -- Access functions and externals in object modules produced by C86 or Lattice C or assembly language. Utilize your existing libraries unchanged!
- Complete Multiple Module Support-- Instant global searches, auto-compile everything that's changed, etc.
- Many more features including batch mode, 8087 support and symbolic debugging.
- Runs on IBM PC, DOS 2.x, 192K and up.
- **Price: $300.00 (Demo $45.00) MC, VISA**

Price of demo includes documentation and shipping within U.S. PA residents add 6% sales tax. Specify C86 or Lattice version.

GIMPEL SOFTWARE

3207 Hogarth Lane • Collegeville, PA 19426
(215) 584-4261

*Trademarks: C86 *(Computer Innovations)*, Lattice *(Lattice Inc.)*, IBM *(IBM Corp.)*, C-terp *(Gimpel Software)*

MYSTIC PASCAL IS 10 TO 1000 TIMES FASTER THAN TURBO

Mystic Pascal compiles at over 100,000 lines per minute on a standard IBM PC. How? When you change a few lines of code, other Pascals make you recompile the whole program—Mystic only recompiles those lines. As if thats not fast enough, the compiler even runs in the background while you are editing. You can recompile a 2000 line program usually in less than one second.

Mystic produces 8086 object code, optimized on two levels. The single precision floating point is 5 to 50 times faster than any other compiler. 4000 multiplications or 2000 divisions per second, without an 8087—compare that to your present Pascal!

Still not sold?! OK. Mystic Pascal is also interactive—Pascal statements can be instantly compiled and executed. And there's a Full Screen Editor. And Help windows for the Standard Pascal language. And support for multi-tasking. And its only—

$39.95!

Requires an IBM PC or true compatible with 256K. Turbo Pascal is a registered trademark of Borland International, Inc.

MYSTIC CANYON SOFTWARE
P.O. Box 1010
Pecos, New Mexico 87552

Place your order today!
Phone or use the coupon!
(505) 988-4214

Name ______________________

Address ______________________

City ______________________

State ____________ Zip ____________

Price is $39.95 plus $4 shipping.
Outside US & Canada add $20 shipping.
Payment must be in US funds on a US bank. Purchase orders accepted from recognized institutions.
NM residents add sales tax.

☐ Check/MO ☐ COD ☐ VISA ☐ MC

Card ______________________

Exp. ______________________

Signature ______________________

CIRCLE 57 ON READER SERVICE CARD

form, the knowledge can be accessed to tell the user why it might be asking for what it is or how it produced the result it did. But for many tasks, there is still no way for expert systems to be used to lighten the workload of human experts. For example, a writer who knew nothing about expert systems could not use a system that is expert on the subject of expert systems to write this article.

DECISION MODELING SOFTWARE

A decision modeling system is one that is intended for problems that involve selecting one of a number of known alternatives. Rather than using knowledge prepared by an application developer, a decision modeling package is designed to allow users to incorporate their own knowledge and opinions in a systematic way to force a conclusion.

Decision Support Software Expert Choice

Decision Support Software Inc.'s system, written in compiled BASIC, allows very complex decision models and provides a graphic representation of the decision's structure. An integral part of the decision-tree display is a powerful tree editor that includes the ability to browse through a decision model at will. There is a complex system of weighting each of the factors and subfactors. All the necessary calculations for distributing and proportioning the weights are done automatically by the system.

To model a decision using Expert Choice, nodes of a tree are created that are represented as boxes extending from a common node above and displayed in a row across the screen as each additional node is added. The root of the whole decision tree is the goal node. Its name is the name of the decision being modeled. Every node box below the goal contains both its name and a number that designates its resultant weight in comparison with its peer nodes on the same level descending from the same parent node.

There is always one node, which is highlighted, that is the current focal node. Arrow keys are used to move about in the tree, thereby changing the node that is highlighted. If the *redraw* command is issued, the display will change to show all the node boxes that are the immediate children of the node that was current when the *redraw* command was issued. When the display of node boxes shifts down the tree, all the upper nodes, including the goal, are represented by oval characters in their proper position, connected by arc lines. The arrow keys can still be used to move to these symbolically represented upper nodes, highlighting them, and to redraw the tree display.

It is easy to move quickly from node to node on the tree display and to redisplay the tree to show important details and to access desired nodes. The *edit* function can be invoked to edit anything in the tree. The nodes affected by the editing operations are the immediate children of the current node when the editor was invoked. The lowermost nodes in any branch of the tree are called the leaves of the tree. Up to seven node children can be created under each parent, and the tree can have as many as seven levels.

From this I think you can see how powerful a tool Expert Choice is for giving a very graphic model of the structure of all the issues and subissues of even the most complex decisions. But even more important than making the overall structure of the decision explicit is defining how the various issues and subissues interact and relate to each other.

To establish how the various criteria or subcriteria relate involves going into the comparison mode. All combinations of two peer nodes on a given level are compared to one another. They are compared on up to three bases: likelihood, preference, and importance. The result of responses to these comparisons assigns relative numerical weights to each of the peer nodes. These relative weights are transmitted proportionally to all of the branches below each node.

Comparisons can be assigned both verbally and numerically. In the verbal mode are multiple choices on how much more or less important, likely, or preferred one issue or criteria is to another. Answers to these questions result in a standard weighting assignment for each degree. In the numerical mode, the user is presented with one screen that has an impact matrix with all of the numerical weightings displayed. Any of the numbers displayed may be changed.

Generally, once you become familiar with how Expert Choice works, it is much faster and can be more accurate to use the numerical mode to assign any values on the one screen rather than to go through all the possible questions in the verbal mode.

Once all the relative weightings for all the criteria have been assigned, it is necessary to make sure that all the main alternatives exist as leaves at the bottom of all the branches you want considered. When the decision model is evaluated, the value for each leaf is calculated. Then the values for all the identical leaves are summed up and the results for each alternative are displayed on a bar graph, usually in order, with the winners highest and the losers lowest.

In a decision model of any size, a potentially formidable editing problem exists in trying to get copies of all the decision alternatives to all the bottom leaf positions where they need to be. Fortunately, Expert Choice's tree editor has a powerful *replicate* command with an option that allows the replication of options to all the leaves of the tree. Thus, very complex and detailed decisions can be modeled with this system. Without this option, the time required to model a decision using this approach would be formidable.

The main limitation of Expert Choice is that it takes far too long for the evaluation of complex decisions because the value of each individual leaf is sent to the screen or printer branch by branch before the final score is displayed. There ought to be an option to see the results as rapidly as they can be computed rather than having to wait. This one flaw detracts from the usefulness of an otherwise very ingeniously designed and welcome package.

Lightyear

Lightyear Inc.'s decision modeling system is very similar in purpose and scope to Expert Choice, with a few pluses and minuses. For instance, Lightyear has no visible tree hierarchy to edit. Instead, everything is done with lists and windows.

The two systems have other differences. In addition to quantified criteria, Lightyear supports a fairly powerful and easy-to-read rule syntax for imposing further conditions on the criteria. Also, the bar chart evaluation of the final results can be displayed at any time without having to wait. You can also ask for detailed evaluation charts of separate criteria in an alternative and have the charts display a comparison of any two alternatives for all criteria. In addition, Lightyear does not limit the alternatives to seven.

From this it may appear that Lightyear is simply a more powerful program than Expert Choice, but Lightyear has its limitations. The biggest of these is that it supports only a single level of criteria, not the numerous levels of subcriteria that Expert Choice allows in its hierarchies. Thus, in spite of its excellent features for limiting criteria and alternatives in complex ways and quickly displaying the results vividly and effectively, Lightyear cannot model decisions that are as complex as those Expert Choice can. So there is a trade-off: either you can choose to model complex decisions with limited options for displaying the results or you can opt for rapid modeling of simple choices with all the bells and whistles for display and evaluation purposes.

Where Lightyear really gets interesting is in areas such as declaring the vocabulary for verbal criteria categories and using rules to limit the role of decision criteria. When a list of decision criteria is prepared, one of three modes must be assigned directly to each criteria. In addition to the verbal and numeric modes found in Expert Choice, there is also a graphic mode. And the modes in Lightyear are not just alternate ways for a user to access any criterion, but categories that are assigned to given criteria according to what is most appropriate for them. Of

Program Editing with

BRIEF™

is More Productive and Less Frustrating
because it will work YOUR way, and BRIEF elegantly integrates:

- A high-level, readable **Macro Programming Language** - allows full parsing or syntax anaylsis. . . . Complete, unlimited variables, etc.
- Edit **multiple** files of **unlimited size** (2 Meg is OK)
- Multiple **Windows** *on screen with different or same file, fragments, etc.*
- A bona-fide **UNDO** stack (up to 300) of **all operations:** deletions, reading files, search, translate, more
- Full **"regular expression search"** - wild cards, complex patterns
- A completely **reconfigurable keyboard**
- **Keystroke macros** - *for common typing sequences*
- Suspend BRIEF to execute, **exit to DOS** - run another program (like a compiler, dir, XREF, DIFF, or DEBUG) then resume BRIEF session
- **Compiler-specific** support like auto indent, syntax check, compile within BRIEF

AVAILABILITY: PCDOS, AT, & Compatibles ONLY $195
Full refund if not satisfied in 30 days
CALL 800-821-2492

Solution Systems™
335-L Washington St., Norwell, MA 02061
617-659-1571

CIRCLE 122 ON READER SERVICE CARD

PROLOG-86™

Become Familiar in One Evening

Thorough tutorials are designed to help learn the PROLOG language quickly. The interactive PROLOG-86 Interpreter gives immediate feedback. In a few hours you will begin to feel comfortable with it. In a few days you are likely to know enough to modify some of the more sophisticated sample programs.

Sample Programs are Included like:

- an EXPERT SYSTEM
- a NATURAL LANGUAGE INTERFACE (it generates a dBASEII "DISPLAY" command)
- a GAME (it takes less than 1 page of PROLOG-86)

PROTOTYPE Ideas and Applications QUICKLY

1 or 2 pages of PROLOG is often equivalent to 10 or 15 pages in "C" or PASCAL. It is a different way of thinking.

Describe the FACTS and RULES without concern for what the computer will have to do. Maybe you will rewrite in another programming language when you are done.

Programming Experience is not required but a logical mind is. PROLOG-86 supports the de facto STANDARD — in "Programming in Prolog" by Clocksin & Mellish.

AVAILABILITY: PROLOG-86 runs on MSDOS, PCDOS or CPM-86 machines. We provide most formats.

Only $125.
Full refund if not satisfied during first 30 days.

Solution Systems

335-L Washington St.,
Norwell, Mass. 02061
617-659-1571
800-821-2492

CIRCLE 42 ON READER SERVICE CARD

course, it all boils down to assigning numerical weights, but to get the best results with a decision model, it is often very important to know how the comparisons are presented.

In the graphic mode, you are shown a sliding scale of the degree to which each decision alternative is affected by a criterion with a graphic mode. To express your grasp of these relationships, you move a marker of each alternative along a linear scale until you find a place that looks like the correct degree of impact. With the numeric mode criteria you supply the weighting factor as a number in the corresponding column.

The verbal mode is more complicated. When you enter the verbal comparison mode you see a rectangular window. On the left are all the criteria that have been designated V, for verbal. The top one will already be highlighted. In the center are the words and on the right the numerical weight values. But the words and numerical weights that appear apply only to the criterion that is highlighted. The words are descriptive terms such as Best, Worst, Average, etc. But these rating terms can be defined as the user sees fit to make them appropriate to the issues to which they refer.

For example, if you were deciding on possible areas for relocating your business, you could define words such as North, South, East, and West for the Area criterion and assign each of them their numerical weight depending on your preference for each. Similarly, as in Expert Choice, you could define the words so that they represent degrees of likelihood or importance or as any basis for comparing the alternatives, such as the degree of known certainty or the degree to which you are sure it is what you want.

One distinctive feature of Lightyear is its ability to further limit and control the decision criteria by defining rules that act as further constraints for the decision. The rule syntax is very easy to read and surprisingly versatile and powerful. A convenient rule editor prepares rules.

There are two basic types of rules to select from: simple rules and *if-then* rules. Once you have chosen your option, a rectangular window is displayed for that rule. You select the criterion involved from a menu and then the operators, such as *MUST BE, SHOULD NOT BE,* and *AT LEAST.* There are facilities for modifying and deleting rules, and the summary evaluation can be displayed immediately to see the affect of the rule on the final outcome of the decision.

Lightyear is a very satisfying program. Its primary advantage is the speed with which an application can be developed and executed. It would work very well with a mouse since it is menu- and list-oriented. One improvement, which would probably make this product the leader in its category, would be adding levels of subcriteria, as in Expert Choice.

EXTENSION PACKAGES

Programming Logic Systems APES

APES—Augmented PROLOG for Expert Systems—is an extension of the PROLOG programming language. This Programming Logic Systems product provides an overlay system for Logic Programming Associate's Micro-PROLOG consisting of several modules that give additional facilities for user interaction and explain the reasoning behind conclusions. This is intended to pave the way for using the augmented PROLOG as an environment for developing and running less user-friendly knowledge systems or for developing a more complete environment using PROLOG for the more familiar type of consultation and rule-based system. In addition, there are modules that assist in editing knowledge bases and in preparing systems with overlay files on disk. This is important because the memory addressing capability of Micro-PROLOG is limited. Ideally, with such an approach and by using a RAM disk, these memory limitations could be overcome and larger applications could be developed.

What APES really offers is a tool kit that helps make the powerful facilities inherent in PROLOG more comprehensible and easier to use. This can be helpful both to developers unfamiliar with PROLOG and for use in building a consulting environment for the end user. The main

Advanced Screen Management made easy

Now a professional software tool from Creative Solutions.

WINDOWS FOR C™

More than a window display system, WINDOWS FOR C is a video tool kit for all screen management tasks.

- Pop-up menus and help files
- Unlimited files and windows
- Instant screen changes
- Complete color control
- Horizontal and vertical scrolling
- Word wrap
- Highlighting
- Auto memory management
- Plus a library of over 50 building block subroutines

Designed for portability. Easy to learn, easy to use.

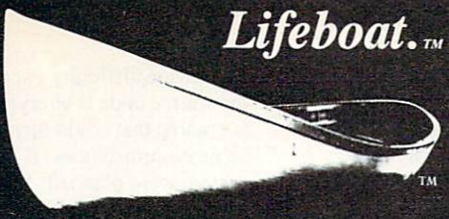
Once you've tried WINDOWS FOR C, you'll wonder how you ever managed without it.

Full support for IBM PC/XT/AT and compatibles, plus interfaces for non-IBM computers; Lattice C, CI-C86, Mark Wm. C, Aztec C, Microsoft C, DeSmet C (PC/MSDOS),

NEW Ver. 3.1
Enhanced portability.
Topview compatible.
WINDOWS FOR C $195
(specify compiler & version)
Demo disk and manual $ 30
(applies toward purchase)
Full source available.
No royalties.

Creative Solutions
21 Elm Ave., Box T7,
Richford, VT 05476
802-848-7738
Master Card & Visa Accepted
Shipping $2.50
VT residents add 4% tax.

CIRCLE 8 ON READER SERVICE CARD



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."

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

CL

CIRCLE 87 ON READER SERVICE CARD

DeSmet C

8086/8088 Development Package $109

FULL DEVELOPMENT PACKAGE
- Full K&R C Compiler
- Assembler, Linker & Librarian
- Full-Screen Editor
- Execution Profiler
- Complete STDIO Library (>120 Func)

Automatic DOS 1.X/2.X SUPPORT

BOTH 8087 AND S/W FLOATING POINT

OVERLAYS

OUTSTANDING PERFORMANCE
- First and Second in AUG '83 BYTE benchmarks

SYMBOLIC DEBUGGER $50

- Examine & change variables by name using C expressions
- Flip between debug and display screen
- Display C source during execution
- Set multiple breakpoints by function or line number

DOS LINK SUPPORT $35

- Uses DOS .OBJ Format
- LINKs with DOS ASM
- Uses Lattice® naming conventions

Check: ☐ Dev. Pkg (109)
☐ Debugger (50)
☐ DOS Link Supt (35)

SHIP TO: ____________________

____

____________________ ZIP ________

C WARE CORPORATION

P.O. BOX C
Sunnyvale, CA 94087
(408) 720-9696

All orders shipped UPS surface on IBM format disks. Shipping included in price. California residents add sales tax. Canada shipping add $5, elsewhere add $15. Checks must be on US Bank and in US Dollars. Call 9 a.m. – 1 p.m. to CHARGE by VISA/MC/AMEX.

Street Address: 505 W. Olive, #767, (94086)

CIRCLE 11 ON READER SERVICE CARD

advantages it provides are:
- Natural language templates for questions posed to the user in a consultation and in answers made to requests for explaining the basis of results
- A menu-generating module
- A facility that can force PROLOG, under instructions, to ask the user for values it needs.

An example of how the PROLOG environment looks when made more friendly by the APES front-end is in Listing 1. Remember that this is not an application program per se, but Micro-PROLOG itself with the APES front-end and utilities added to it! In this context, a user could delay responses to the "Answer is" prompt by asking for an explanation or making a sophisticated query to the PROLOG data base.

What APES points out most clearly is the power that PROLOG offers for expert system development. By adding a more English-like front-end, an explanation facility, and various utilities and using no inference procedure or representation language other than PROLOG, environments can be built that are suitable as is for many knowledge system projects and that can be extended for other systems.

The one difficulty I see is that because the source code is encrypted, a black box is created that could turn into a liability in some circumstances. If serious use of this system were planned, I would suggest that some arrangement be made with the authors to purchase a source code license. Although this could be expensive, it might prove economical in comparison with unforeseen development costs.

Mountain View Press Expert-2

Forth programmers have the opportunity to learn how a simple rule-based reasoning system is built with this package written by Jack Park and distributed by Mountain View Press.

Expert-2 comes on two disks, one with a run-time knowledge system module and the other with Forth source code screens. The user's guide contains a tutorial on expert systems that can help the complete novice grasp some of the fundamentals in knowledge system processing. Also included are tutorials that take beginners through the steps involved in writing programs in both Expert-2 and Forth. An example of Expert-2's rule syntax is presented in Listing 2.

```
Which fruits did Peter like?
Choose from the following
1 -    apples
2 -    pears
3 -    oranges
4 -    bananas

Answer is pears                   (an earlier answer)
Answer is 1                       (selecting apples)
Answer is 2                       (selecting pears)
** I already know Peter likes pears
Answer is oranges                 (selecting oranges)
Answer is end
```

Listing 1.

```
    ( RULE 1 -- Measles )
    IF subject has rash on scalp, then body
    AND subject has brownish pink rash
    AND subject has white spots inside cheek
    AND subject has conjunctivitis
    AND subject has bloodshot eyes
    AND subject has harsh hacking cough
    AND subject has high, fast, rising fever
    AND subject has runny nose
    THENHYP is Measles
```

Listing 2.

In this scheme, capitalized words are operator key words. Expert-2 has five main types of operators: basic operators, negative context operators, *RUN*-type operators, the *THENHYP* operator, and *BECAUSE* operators. The basic operators are the *IF, AND,* and *THEN* familiar in all production rule languages. The negative context operators are *IFNOT* and *ANDNOT*. An *IFNOT* statement is used as a condition that stipulates what will follow in the event that a state of affairs is not the case. *ANDNOT* operates like *AND* in stipulating additional conditions, except that the described situation is not true.

RUN-type operators are used to access Forth subroutines. They include *IFRUN, ANDRUN, THENRUN,* and *ANDTHENRUN*. They all work analogously to their namesakes, but instead of referencing a statement string they reference any Forth word that has been defined and loaded. The one stipulation is that the words for the subroutines have to be loaded before the rules. The *THENHYP* operator states that the string that follows it will be considered a hypothesis to be tested rather than an outright truth. Finally, *BECAUSE* and *BECAUSERUN* are keywords that stipulate how the system will respond to a why (W) request by the user. With *BECAUSE* you can provide a statement like the others in a rule. *BECAUSERUN* allows you to call your own Forth word that can do whatever you want it to do in generating an elaborate explanation.

Like the representation scheme, the inference mechanism is extremely simple. It seems to have been derived from an example in the first edition of Patrick Winston and Berthold Horn's *LISP* (Addison-Wesley). The main word is *DIAGNOSE*, which calls on *VERIFY*. *DIAGNOSE* accesses *HYPSTACK*, the hypothesis stack, and gets the pointer to the character string that is the hypothesis it will try to prove. If the hypothesis is found to be true, *DIAGNOSE* stores its pointer in *CURHYP*. If not, it fetches another pointer from *HYPSTACK*. When *VERIFY* is called, it picks up a duplicate of this hypothesis pointer and attempts to see if the hypothesis is true. To do this, it calls on *RECALL* to see if the pointer is among those things considered facts.

One interesting side note to Forth aficionados is the use of a forward reference in this program. When *TESTIF+* calls *VERIFY*, it uses a feature of MVP-Forth that allows duplicate names by using a version of *VERIFY* that is actually an execution vector. Normally in Forth you cannot call a word until it has already been defined.

Although Expert-2 is an extremely simple program and intended only as a learning aid, with the inclusion of the source code and the *RUN*-type operators, programmers should note that the full Forth language is present. Thus they can extend the program, write other inference engines for it, and add their own Forth subroutines freely from rule sets. A start on a forward-chaining inference procedure is suggested with *FINDRULES >*.

This is a nice, clearly explained package for programmers to experiment both with simple knowledge engineering concepts and the implementation of rule processing and reasoning environments.

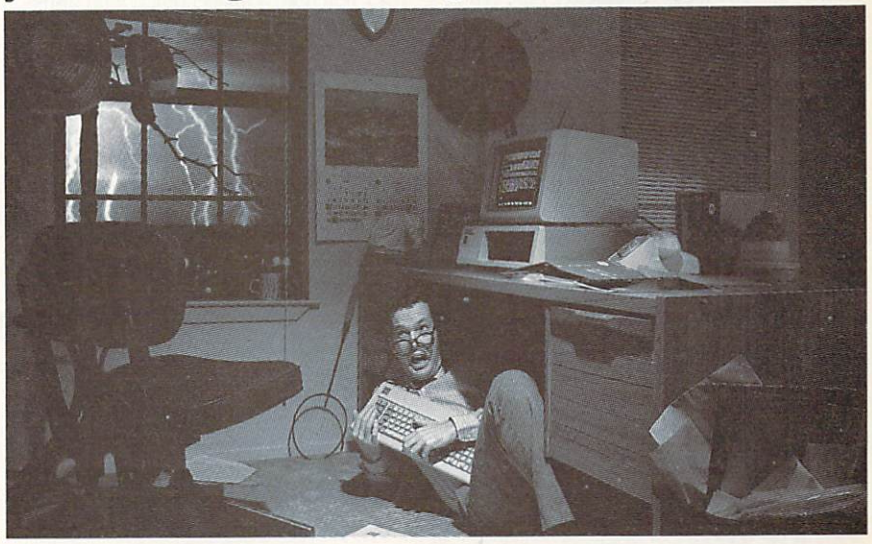
RULE INDUCTION SYSTEMS

Expert Systems Expert Ease

It was inevitable that products offering ease of use in exchange for limited capability would emerge. Here the attraction is a decision tree system that has a spreadsheetlike editor and is totally example-driven. No rules have to be written with Expert Systems Inc.'s Expert Ease. Specific examples are entered and a type of decision rule is inferred that will handle the examples and other similar cases. While this sounds very easy, early reviews criticized Expert Ease for not being that easy to use at all, and I have to admit that when I first tried it, I was a little puzzled by exactly how everything fell into place.

Some critics of Expert Ease have claimed that its knowledge bases consist of only one large rule. However, it might

If lightning still scares you, you're using the wrong file manager.



Be sure. Btrieve™.

Lightning may strike. But it doesn't have to destroy your database.

Btrieve™ file management offers automatic file recovery after a system crash. So accidents and power failures don't turn into database disasters. Your Btrieve-based applications will come up when the lights come back on.

Fast. Btrieve is lightning fast, too. It's written in Assembly language especially for the IBM PC™. And based on the b-tree file indexing system, with automatic balancing and electrifying access speed.

The standard for networking. Btrieve/N (network version) sets the standard for the industry's most popular LANs, including IBM's PC Network.

Fully-relational data management. SoftCraft's entire family of products gives you a complete, fully relational database management system. Rtrieve™ adds report writing capabilities. Xtrieve™ speeds users through database queries with interactive menus.

For professional programmers. Btrieve is the fast, reliable answer for all your application development. In any development language—BASIC, Pascal, Cobol, C, Fortran, and APL. With Btrieve, you can develop better applications faster. And know they'll be safe if lightning strikes.

SoftCraft Inc.

P.O. Box 9802 #917 Austin, Texas 78766
(512) 346-8380 Telex 358 200

Suggested retail prices: Btrieve, $245; Btrieve/N, $595; Xtrieve, $195; Xtrieve/N, $395; Rtrieve, $85; Rtrieve/N, $175. Requires PC-DOS or MS™-DOS 1.X, 2.X, or 3.X. Btrieve, Xtrieve, and Rtrieve; IBM; and MS are trademarks of SoftCraft Inc.; International Business Machines; and Microsoft Inc.

CIRCLE 31 ON READER SERVICE CARD

be more accurate to say that the cases entered cannot lead to a rule that draws any conclusion other than one of the possible final outcomes. This is equivalent to a system where various rules are possible but where the *THEN* part must always be about the final outcome rather than an intermediate result.

Though this is true of Expert Ease, it is not a final limitation of the system because a chaining capability allows the conclusion of a decision tree to result in loading a new knowledge base beginning at the top of a new decision tree. In this way, a network of decisions can be built where only the final leaf evaluated represents the outcome. The slowdown in speed caused by having to load another file from the disk for each link of the chain can be eliminated by using a RAM disk. Still, this does not put Expert Ease on an equal footing with systems that can simply use any legal statement in the representation language in the *THEN* part of a rule. The construction and debugging of these decision tree chaining networks is not a trivial task and is more difficult than simply adding rules in a production rule system.

An application is developed in Expert Ease by making various entries in several different screens in the spreadsheetlike display. Below the screen display is a command line menu listing various single character commands. First enter "a" for the attribute screen where the main factors in the decision are declared. The attributes are one of two types, integer or logical. The default is integer. To convert this to logical, a first value must be entered from the attribute screen as a string. The attribute is then displayed as the logical type.

To begin entering example values for the attributes, shift to the examples screen. Already a number of flaws are apparent in Expert Ease's user interface that make it more difficult to use than it should be. All entering of values should be from one mode screen, and there should be an explicit command on the attributes screen for declaring attributes either integer or logical.

The design has other faults also. First, there is generally no consistent way to back up once you've selected an editing command but then decide that's not what you really want to do. For example, in using the *change* option. I found that rather than being able to hit Escape to back out, I had to enter false values to get back to the editing menu and then delete the false values. Also, commands like *n* and *v* must be reentered each time a new entry is made. These should be offered as modes that can be toggled so that in entering a list of attributes or values the command does not have to be issued separately each time. Another difficulty with Expert Ease is that on-screen help, although available from the main menu, often is not available in situations where the user actually needs it the most.

When all of the attributes and values for a decision are entered, the next step is to use them to generate a rule. This is done simply by hitting the ! character. The rule that has been induced can then be inspected on the rule screen.

After the rules have been induced, the next step is to test out the decision by selecting the query mode. Left to itself, Expert Ease will generate automatic questions that ask for the value of each attribute and offer multiple choices that are selected by number. Fortunately, a text option allows you to word your own questions as well as the conclusion.

Perhaps the most important thing to recognize about Expert Ease is that there is no facility for asking how a conclusion was reached. This is not a trivial omission. One of the most important differences between rule-based systems and knowledge-embedded programs written with conventional algorithms is that because the knowledge is represented explicitly in rules, it can be accessed to help the user see the reasoning involved so that he or she is not expected to accept the conclusions blindly. For this reason, Expert Ease is not appropriate for any important applications where it would be irresponsible not to evaluate and question the result.before making an independent decision.

In addition to Expert Systems Inc., a company called Human Edge Inc. is also liscensed to market Expert Ease. Human Edge is located at 2445 Faber Pl., Palo Alto, Calif., 94303, (415) 493-1593.

KDS

If talking to the outside world by real-time data acquisition or the ability to call external programs is a central issue, then the KDS system may be worth looking into. This KDS Corp. product is an example-driven rule induction program designed along the same lines as Expert Ease.

KDS, though, has a number of advantages over Expert Ease. First of all, it runs under MS-DOS rather than the UCSD p-system, which means that it can and does interface with other MS-DOS programs. In addition to this, it provides for certainty factors and has two alternate inference options and a back-up key that allows both a user and a developer to back their way through the decision tree at any point. Also, if you like skillfully designed color displays, then you may find the use of color text displays in KDS very effective.

Another important factor is KDS's speed. Written entirely in assembly language, it is very possibly the fastest of all the systems reviewed here. I also think that KDS is somewhat easier to use than Expert Ease. The right screen panel is devoted to on-screen help messages that shadow where you are in the system and tell you what you need to know to proceed. It also allows considerably larger applications to be developed. The Playback module for end users can be purchased for $495.

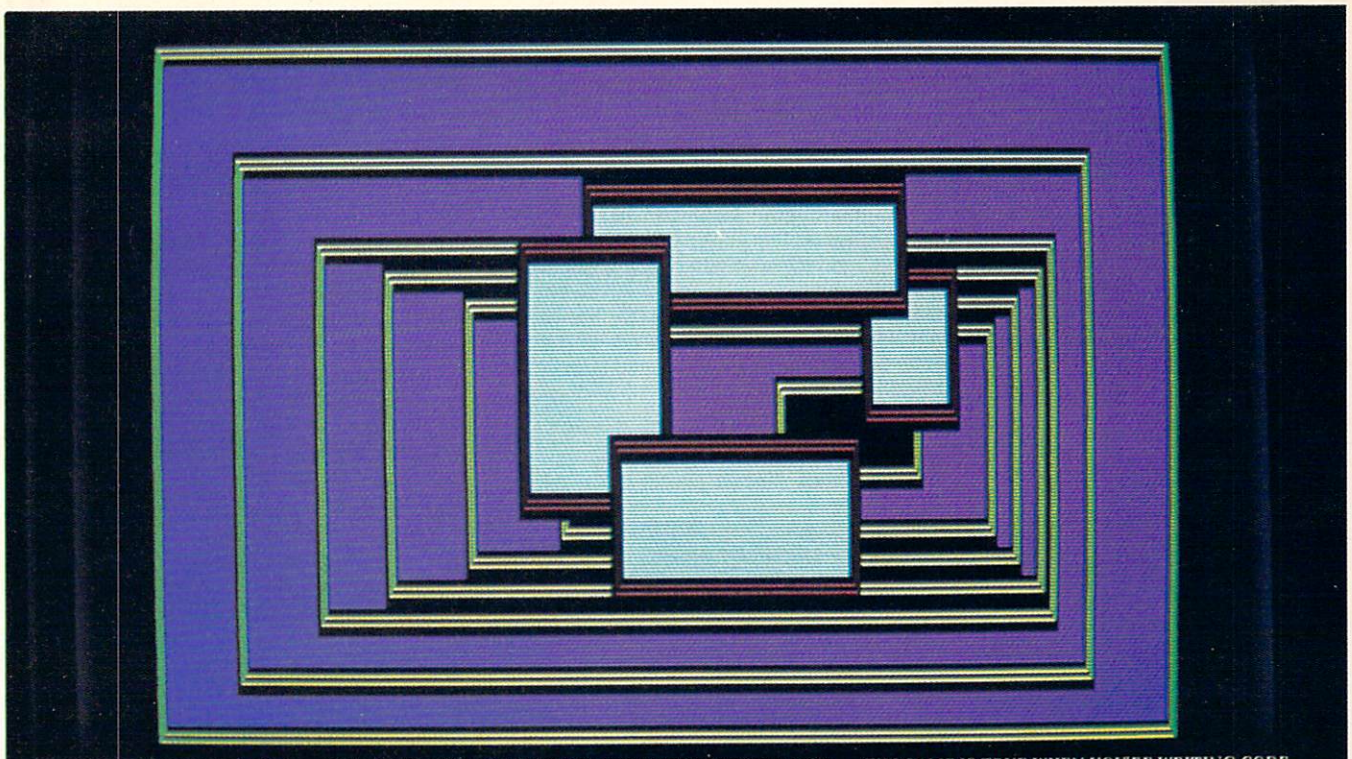
KDS does not deal with attributes and values but rather conditions described in text strings that are either true or false or known with some degree of certainty. One advantage of this is that it is not restricted to just one category of results for a given knowledge module. But it also means that the system acts like it creates only binary trees and the user is answering only yes or no to each condition. Actually, KDS is somewhat better than a binary decision tree system. To see how, it's best to know something about how the development environment works.

When you first boot up KDS, you see this menu:

```
[1] Enter new cases
[2] Edit
[3] Transform to finished KMOD
[4] Make rules
[5] Print-outs
[6] Initialize knowledge file
[7] Edit reference numbers
[8] Require a definitive answer
[9] Quit
[F1] Display case conclusions
[F2] Display conditions
```

To get started on a new application, you would hit [6]. The prompts for file name, author, copyright notice, etc., then appear. Next you are asked for the first condition. Then you see two more prompts, one for a conclusion that follows if it is true and another if it is false. After these are entered you are returned to the menu. Now you can choose [1] to add further cases. The conclusion of each case is a separate outcome and consists of whatever text it is that you type. This means that there is no way for KDS to keep track of which conclusions are about the same issue or alternate options. There's just a string of text that has a certain place in the hierarchy, along with some factors associated with it.

Many advanced features of KDS give the developer some powerful and convenient tools for chaining applications by creating additional files on subdirectories of the data disk. The KDS manual, however, is quite ill-suited to this otherwise very streamlined product. It is rambling, verbose, and poorly conceived and organized. A product like this needs a quick guide to get the end user up and running as quickly as possible, a well-organized reference guide to let the experienced professional see quickly everything that is there, and a knowledge system development tutorial for complete novices.



WHEN YOU BUILD A HOUSE . . . YOU DON'T NEED TO MAKE THE WINDOWS YOURSELF. NOW . . . THE SAME IS TRUE WHEN YOU'RE WRITING CODE.

Why did Simon & Schuster, 3Com, Tymshare, and Revlon choose VSI–The Window Machine?

(and how come you can buy it for such a low price?)

$59.95

Windows With A View Toward The Future

The Window Machine™ occupies only 12K! Written in tight, fast Assembler, it performs like a racing engine . . . with more power than you'll probably ever need. Yet, it's an engine designed to fit in the vehicle of your choice . . . from a "stripped-down" 128K IBM PC to a fully loaded AT. The programs you write today will run on the broadest range of machines possible . . . now, and in the future.

These are coders' windows . . . designed to be built into the programs you are writing. They can overlap, move anywhere on the screen, grow, shrink, vanish or blink. They can be bordered in anything from a simple line to flashing asterisks . . . or even no border at all. And you can have up to 255 of them at a time! Color or monochrome . . . of course!

Windows Bigger Than Your Screen?

Here's where the VSI part of our name fits in. VSI means Virtual Screen Interface. Behind each window, there's a much bigger picture. VSI defines virtual screens rather than just windows. The window itself shows whatever portion of its virtual screen you wish to exhibit at any given point in your program. Each screen can be up to 128 x 255 (columns x rows, or rows x columns). And there are more than 100 screen primitives at your command.

Multilingual Windows

You can order The Window Machine with the language interface of your choice: C, Pascal, Compiled Basic, Fortran, Cobol, or PL1. We've even recently completed an interface for Turbo Pascal*, so that now true, full-featured windowing can be utilized with this fine compiler. (Turbo's own built-in "windowing" procedure is extremely limited).

Windows That Won't Break You

We decided to save you a lot of money. So, we left behind fancy binders, monogrammed slip cases and plastic presentation boxes. Instead, you'll find an extremely powerful tool and a 200 page manual written with an eye toward simplicity, clarity and completeness. (We figured if you wanted ribbons and bows you could always add them yourself.)

And by offering you the product ourselves, we were able to cut out all the middlemen and save you a tremendous amount of money.

**Turbo Pascal is a Trademark of Borland International*

VSI THE WINDOW MACHINE

Available for the IBM PC, XT, AT, IBM Compatibles, Wang, T.I., and HP 150

The Window Machine Includes:

- *Zoom Windows*
- *Multiple Virtual Screens (up to 255)*
- *Choice of Borders (including flashing borders)*
- *Support for all Color and Monochrome Video Attributes (no graphics card required)*
- *Built-in Diagnostics*
- *And much, much more*

ORDER YOUR COPY OF VSI—THE WINDOW MACHINE TODAY
For Visa & MasterCard orders call toll free:
800-538-8157 Ext. 824 In CA 800-672-3470 Ext. 824
Call Mon.-Fri. 6A.M. to 12P.M., Sat. & Sun. 6A.M. to 8P.M. (P.S.T.)

The Window Machine™ $59.95 + $5 Shipping and Handling

LANGUAGE INTERFACE:

☐ Lattice C ☐Realia Cobol ☐Microsoft Basic Compiler ☐ Microsoft Fortran
☐ PL1 ☐Microsoft Pascal ☐Turbo Pascal (full featured true windowing)

COMPUTER ____________________

Name ____________________

Address ____________________

City ____________________ State ____ Zip Code ________

☐ Check ☐Money Order ☐VISA ☐MasterCard

Card # ____________________ Exp. Date ________

*California residents: tax included. Orders outside USA: Please add $10 for shipping and handling

30 day Money Back Guarantee

AMBER
AMBER SYSTEMS
1171 S. Saratoga-Sunnyvale Road
San Jose, CA 95129

CL

AMBER SYSTEMS, INC. 1171 S. Saratoga-Sunnyvale Road, San Jose CA 95129

FOR DEALER INQUIRIES: CALL OUR 800 NUMBER

CIRCLE 2 ON READER SERVICE CARD

SMALL PRODUCTION SYSTEM TOOLS

Level Five Research Inc. Insight Knowledge System

The only way to begin a description of Insight is to say that it is the best bargain around for the complete novice who wants to dabble in knowledge system development to find out what it is all about. This system, though in no sense the most powerful in the lineup, is very nearly foolproof. Once you understand how the system works, you can go about making running knowledge systems that work without a lot of syntax problems and other complexities that distract you from concentrating on the problem your application is supposed to help solve.

Insight is both user friendly and fast. A lot of care has gone into designing its user interface. The system is entirely driven by function keys and menus and deliberately refrains from supporting color displays, so that those who have color monitors see everything in dull, unadorned white.

As you proceed through a consultation with Insight, you are presented at each step with a distinct set of choices selected by indicated function keys that vary depending on the options chosen for the user. At each step, your options are kept at a minimum and carefully laid out for you, making Insight very easy to learn and use.

The other components of Insight are the rule language and the compiler. To prepare a knowledge base, an external text editor like Volkswriter Deluxe or the EC editor can be used. When you want to test the knowledge base, you compile it with a separate compile program and run it in the consulting component.

The representation language is very rudimentary. Inequality operators are available for comparing numbers, but there are no numeric variables and no arithmetic operators for making simple calculations. Confidence values, however, are supported.

The inference engine employed by Insight is a simple backward chaining type of the familiar EMYCIN variety. It incorporates a structured goal mechanism that allows the prime interest areas of a problem to be specified. The system comes with an easy-to-read, 44-page manual that is sparse on details about the inner workings of Insight. As a system intended primarily for the student of knowledge system development, a more comprehensive user guide and tutorial would be far more appropriate.

EXSYS

A little more expensive than Insight, but with substantial additional power, EXSYS Inc.'s product offers calculated variables, a rule editor, color displays, and the ability to execute external programs that can be written in your favorite programming language. If your main objective is simply to get acquainted with the knowledge engineering process or to do some quick prototyping or proof of concept work for a proposed expert system, then EXSYS would be a relatively inexpensive way to go.

This system has a number of convenient features. The representation language is surprisingly powerful. Compared with the more advanced and higher priced systems, EXSYS lacks some important things, but it's a good idea to see how far you can take a system like this. By using a few tricks, you'll find ways to get around features you thought were necessary but are absent in EXSYS.

The system consists of two main programs: EXSYS.EXE, the run-time environment, and EDITXS.EXE, the development environment. One of the nice things about the EXSYS rule editor is the way the conditional qualifiers and choices, or outcomes, are kept in numbered lists so that it is not necessary to repeatedly retype the same items in different combinations.

The main editing screen is divided into three windows, a right and left screen, and a command line at the bottom. The choices or outcomes are automatically

LATTICE® WORKS

LATTICE TOPVIEW TOOLBASKET RELEASED

If you develop programs oriented to take advantage of IBM's TopView multi-tasking window environment, you need the Lattice TopView Toolbasket.

The Lattice TopView Toolbasket is a library of more than 70 C functions to control window, cursor, and pointer functions, along with printer controls, cut and paste functions, debugging, and general utilities. It also includes an assembler interface and master file and data definition headers.

This new Lattice product speeds your program development with its documented tips on handling I/O and dispatch routines, plus its sample programs with source code and batch files your programs can be patterned after.

The Lattice TopView Toolbasket runs on the IBM PC, XT, AT, and compatible systems with 256K (512K and TopView Toolkit from IBM recommended). The Lattice TopView Toolbasket is available for $250. Binary *and* Source Code available for $500. The Lattice TopView Toolbasket was developed for Lattice by Strawberry Software.

Speed your TopView program's release. Order the Lattice TopView Toolbasket today!

UNICALC COMPONENT SPREADSHEET SOURCE KIT CHOSEN BY ORACLE

Lattice, Inc. and Oracle Corp. have jointly developed SQL-Calc, a unique product based on Lattice's UNICALC spreadsheet program and the ORACLE Database software.

SQL-Calc gives users the ability to extract and manipulate database information using spreadsheet features made popular by programs such as VisiCalc, SuperCalc, and Lotus 1-2-3. SQL-Calc is the first full-featured spreadsheet to be coupled so closely with a powerful relational database to give users the ability to treat SQL database access statements like normal spreadsheet formulas.

The UNICALC Source Kit, used to develop SQL-Calc, is a component electronic spreadsheet kit for UNIX, MS-DOS, PC-DOS, and other systems that support the C language, and is available to other program developers. It allows program developers to incorporate electronic spreadsheet features into new or existing software programs. UNICALC also gives users many display options and will generate printed reports in a variety of formats which may be saved for further manipulation by a word processing system.

The UNICALC Source Kit is available with a wide range of royalty and licensing arrangements at very competitive pricing. Call Lattice, Inc. at (312) 858-7950 to discuss your applications.

LATTICE C COMPILERS CHOSEN BY MORE THAN 26,000 SOFTWARE DEVELOPERS!

The top-selling C compiler, Lattice C, is now published directly by Lattice, Inc.

When you purchase our edition, you get support directly from the people who wrote the C compiler. You also get free "bug fix" updates during the warranty period. When you register your purchase with us, you are notified of all updates, enhancements, and new C programming tools as they become available. And, as with all Lattice products, you are covered by a money-back guarantee.

When you are ready to purchase a C compiler, consider the source. Then call us: Lattice, Inc.

ASK ABOUT OUR "TRADE UP TO LATTICE C POLICY"

Lattice, Inc.
P.O. Box 3072
Glen Ellyn, IL 60138
(312) 858-7950
TWX 910-291-2190

International Sales Offices
Belgium: Softshop. Phone: (32) 53-664875
England: Roundhill. Phone: (0672) 54675
Japan: Lifeboat . Phone: (03) 293-4711

CIRCLE 18 ON READER SERVICE CARD

numbered so that to enter one from the list into a given rule it is only necessary to type its number. The same is true of the conditions as well as each condition's list of values. In each case, after you have entered a factor once, it is simply a matter of selecting numbers, and the corresponding text is automatically entered in the rule. If at some point you decide that you want to rearrange the order of the rules, there is a function for doing this.

Mathematical formulas are not handled in the same way. They must be retyped each time. Another peculiarity of the arithmetic functions is that the numerical variable names, unlike the qualifiers and choices, cannot be deleted altogether from the system once entered. This is a minor flaw, but it limits your ability to experiment with things you might not want in the final version.

There are three main modes for encoding an EXSYS knowledge base: for yes or no answers only, for certainty factors ranging from one to 10, and for certainty factors ranging from one to 100. Once an option is chosen it will hold the entire knowledge base. EXSYS also has options regarding such things as whether rules should be checked for consistency, whether rules should be displayed as rules to the end user, and whether variables should be displayed at the end of a session.

On the whole, the EXSYS rule editor provides an extremely convenient environment for development. The one drawback of this type of system is that EXSYS writes its own file formats that do not allow for editing rules in a normal text editor. Also, creating backups is extremely important, because if for any reason your knowledge base file becomes damaged and the EXSYS environment cannot read it properly, you have no other recourse.

EXSYS also has the ability to call external programs. So far this can be used in a limited way to get additional data, to perform calculations, and to make the data available to an EXSYS application. This can be done in two different ways. One method is used when just one value needs to be returned. The other enables a number of different values for variables to be made accessible to an application running in EXSYS. For returning single numerical values, the *RUN* procedure can be called from a rule to execute a program that requests input for calculating a value that will be made accessible to EXSYS by creating the file RETURN.DAT, which EXSYS reads automatically.

To return multiple variables as well as qualifiers involves making a special declaration to the rule editor at the outset. Also, a special format must be used in the RETURN.DAT file so that, in addition to the value itself, EXSYS knows whether it is a variable or qualifier and what its number is. Using this option, programs that read data base or spreadsheet files and then write the values in the format to be picked up by the EXSYS knowledge base can be called from EXSYS.

The one major limitation of both these procedures for interfacing with external data and programs is that there is not yet a way for EXSYS to pass data to the external program. However, plans exist to update this in the future, and it appears that this program, which was quite rudimentary when first released, will continue to add the capabilities that make it an increasingly attractive option for many applications.

EXSYS already is being applied to a number of interesting uses. It is being used by the U.S. Dept. of Agriculture to aid farmers in deciding when and how much to irrigate crops, by a railroad company to rapidly troubleshoot equipment malfunctions automatically, and by the U.S. Dept. of Energy to aid in decisions about security classification of information.

Expert Systems products and manufacturers

Expert Choice—$495
Decision Support Software Inc.
1300 Vincent Place
McLean, Va. 22101
(703) 442-7900
(800) 368-2022 (orders only)

EXSYS—$295
EXSYS Inc.
P.O. Box 75158
Contract Station 14
Albuquerque, N.M. 87194
(505) 836-6676

Expert Ease (v 1.1)—$475
Expert Systems Inc.
868 West End Ave., Ste. 3A
New York, N.Y. 10025
(212) 662-7206

KDS—$795 (development system)
KDS Corp.
934 Hunter Rd.
Wilmette, Ill. 60091
(312) 251-2621

Insight Knowledge System (v 1.2)—$95
Level Five Research Inc.
4980 South A-1-A
Melbourne Beach, Fla. 32951
(305) 729-9046

Lightyear—$495
Lightyear Inc.
1333 Lawrence Expwy., Bldg. 210
Santa Clara, Calif. 95051
(408) 985-8811

Expert-2—$100
Mountain View Press Inc.
P.O. Box 4656
Mountain View, Calif. 94040
(415) 961-4103

APES—$395
Programming Logic Systems
31 Crescent Dr.
Milford, Conn. 06460
(203) 877-7988

ACTIVE TRACE

"Software that lives up to its promises. *When a Basic program doesn't work the way you want it to, this package... will help you track the problem down... Scope is a tool for the beginning, advanced, or professional programmer, and it begins where the cross reference maps leave off."*
Howard Glosser, *Softalk for the IBM Personal Computer* July '84, pp 120-121

"Extremely useful program... *Anyone doing much programming in Basic should appreciate Active Trace a lot."*
Jerry Pournelle, *Byte Magazine* April '83, p 234

"A marvelous Basic programming aid... *It's just amazing to watch a program you wrote run under Scope, and debugging becomes if not trivial, then at least doable"*
Thomas Bonoma, *Microcomputing,* Dec. '83, p 22

*"...**a really neat utility**... designed to untangle even the most convoluted Basic program.... The documentation is almost worth the price of the package."*
Susan Glinert-Cole, *Creative Computing,* July '84, p 210

Active Trace will lead you through your program letting you know variable values (all variables or just those you specify) as they change. Your program's internal activity is presented on your screen, or printer, or it can be saved on disk. **It's simple, effective and works with the BASIC you already own.**

Active Trace $79.95
Includes Scope, XREF mapping and documentation

Active Trace is available for most **MS-DOS** and **CPM 2.2** systems and supports the special features of Brand specific versions of Microsoft Basic such as **Basica** on the **IBM-PC**.

AWARECO
Active Software
P.O. Box 695 Gualala, CA 95445
(707) 884-4019
800-358-9120(US) 800-862-4948(CA)

Active trace, Active software, and Scope are trademarks of AWARECO—CPM is a trademark of Digital Research—MS-DOS and Microsoft are trademarks of Microsoft Corporation—IBM-PC is a trademark of IBM Corp.

CIRCLE 5 ON READER SERVICE CARD

Microprocessor programming made simple.

"Keep it simple" was the principle of the 14th Century English philosopher William of Occam and it has even more validity today. Faced with the problems of sophisticated computer systems, designers have found that ever more complex programming languages are further complicating their tasks. Until now.

Occam. Created for system design and implementation.
When we started designing our new VLSI family of 10-MIP transputers, we built on William's simple philosophy. To take advantage of the possibilities opened up by the transputer, we needed to create a language capable of properly addressing parallelism and multiprocessor systems.

With the ability to describe concurrency (whether timeshared or real) and to handle message-passing at the lowest level of the language, all aspects of a system can be described, designed and implemented in occam. From interrupt handling through signal processing to screen editors to artificial intelligence. And on.

But occam is not limited to our transputer family. It provides an efficient, responsive implementation language for systems built on today's microprocessors. It also opens up future possibilities with its performance-enhancing multiprocessor capabilities. And INMOS now offers a product to let you exploit occam's total capability in your system.

Simplify your job with the Occam Programming System.
The Occam Programming System (OPS) gives you the tools for complete VAX / VMS software development. This package includes an integrated editor / checker, an optimizing VAX compiler and full documentation. This gives you a supportive environment for the development of occam programs for execution on the VAX. Cross-compilers for 68000 and 8086-based systems will also be available.

What's more, the occam programs developed and proven on the OPS will give you a head start for work with the INMOS transputer. Extensions to the OPS will be available which will allow occam programs to run on the transputer.

And if you have a requirement to program the transputer in other popular high-level languages, other extensions will include compilers for C, Fortran, and Pascal.

Get started today.
Contact us for our information pack on occam, the Occam Programming System and the transputer. You'll be surprised how simple your life can be.

For quick response, call us at (303) 630-4000 or write:
Occam, P.O. Box 16000,
Colorado Springs, CO 80935.

inmos

inmos, and occam are trademarks of the Inmos Group of Companies

CIRCLE 32 ON READER SERVICE CARD

Shopping for a LISP

By Tom Kenyon

LISP lives on. Most of the languages developed in the late 1950s and early 1960s have long since fallen by the wayside (FORTRAN is one great exception). But LISP continues to survive and even grow in strength and capabilities, and the recent swell of interest in artificial intelligence-based applications has not diminished this growth. In fact, AI has become primarily responsible for LISP's continued existence.

LISP is not the sort of thing that readily lends itself to a comparative style of review. By its very nature, LISP is a beast that seems to defy quick and easy descriptions, being as much a mind-set as a language. But how is one to critique the uniqueness of mind-sets? Granted, the standard benchmarks can be run. Feature matrices can be constructed. Commentary can be written. Yet none of these can fully convey the feel of these products.

And the feel is what makes the difference. To a large degree, this feel is created by the environment offered, which is a result of integrating the editor, interpreter, debugger, and (in the case of a few) compiler, plus the style and quality of the documentation. It is the very seamlessness of this integration that imparts a distinct ambiance nearly impossible to capture in the nonpoetic prose of a comparative review.

Each of these products had a different feel, and it would be difficult to say that any one was better than the others. But one attribute shared by all these products is the sense of satisfaction their authors must feel in providing realistic, workable, symbolic programming to the micro environment. It was a joy to work with these LISPs and talk to their authors. Heroes are hard to find.

Test methodologies. There is a danger in relying too much on the results of benchmark tests. LISP, unlike C or Pascal, is not a language that tells its full story via execution speed. It should be emphasized that most LISPs are environments for symbolic manipulation and not just test beds for number crunching.

The diversity of the dialects encountered in these tests, as well as the sheer size of some of the packages, preclude the authors from certifying that all efforts were made to optimize the benchmark code for the particular package tested. My guess would be that almost all of the benchmark figures could be improved upon after some experience with the package.

All three benchmarks required high degrees of recursion. This was intentional because LISP by nature is a highly recursive language. Two of the benchmarks are standard for almost any language: Factorial and Fibonacci (Listings 1 and 2). The third, MYREV/APP (Listing 3), is specialized for LISP and is a LISP implementation of the standard LISP primative functions *REVERSE* and *APPEND*. This combined function was intended to give the symbol-slinging aspect of the language as hard a workout as possible.

The MS-DOS benchmarks (Table 1) were conducted on a Leading Edge PC set with a clock speed of 7.14 MHz. All of the benchmark figures are in the format MINUTES:SECONDS.-HUNDREDTH-SECONDS. Table 2 presents a useful general criteria cross-reference for each package.

Levien Instrument BYSO LISP

BYSO LISP's young creator, Raphael Levien, is to be congratulated for this relatively fast and complete package. BYSO is not without its problems but should overcome these as it matures.

BYSO has the standard complement of LISP functions but is not too well documented. The writing in the manual is stilted and, to some extent, incomprehensible. The section on the editor still has me puzzled.

MS-DOS benchmarks

Manufacturer and product	Factorial	Fibonacci	MYREV/APP
Levien Instrument BYSO LISP	—	9.01	2:28.41
ExperTelligence Inc. ExperLISP	0.79	0.32	—
Gold Hill GCLISP	—	11.20	1:23.26
Integral Quality IQLISP	3.74	16.28	2:30.88
Software Toolworks LISP/80			
Norell Data Systems LISP/88	—	2:23.21	16:05.48
Microsoft MuLISP	3.11	5.31	29.64
The Lisp Co. TLC-LISP (compiled)	—	9.37	56.15
The Lisp Co. TLC-LISP (interpreted)	—	23.73	2:45.66
Northwest Computer Algorithms UO-LISP (compiled)	0.97	2.81	10.37
Northwest Computer Algorithms UO-LISP (interpreted)	6.58	31.44	10:10.02
ProCode International Waltz LISP	2.26	13.17	1:23.26
XLISP	—	1:46.04	11:11.32

Table 1.

BYSO does not deal well with a color graphics adapter. Each keystroke generated a burst of color and snow across the screen. Suspecting that IBM compatibility problems existed with the Leading Edge PC the tests were conducted on, I ran BYSO on an associate's straight IBM system and encountered the same problem.

BYSO also did not deal well with buggy code. Rather than trapping errors and reporting them, BYSO had an annoying habit of going off into never-never land and remaining there until the computer was switched off and back on.

When things were working well (on a monochrome monitor with perfect code), BYSO performed admirably. The benchmark times were quite respectable, as was the completeness of the implementation. Unfortunately, BYSO's negatives outweigh its positives. At $125, BYSO does not constitute a good buy. The icing on this none-too-tasty cake is that BYSO is copy protected, bad news for a language product. For a few dollars more, one could purchase Waltz LISP, UO-LISP, or IQLISP, all of which perform well and are adequately documented.

Gold Hill Computers GCLISP

GCLISP (Golden Common LISP) is a subset of what many think will become the de facto standard LISP dialect—Common LISP, which is big, full-featured, and complex. But GCLISP differs from Common LISP in one major respect—scoping. Common LISP is lexically scoped, while GCLISP is dynamically scoped. This presents a major compatibility problem. Gold Hill claims operational efficiency as the reason for dynamically scoping GCLISP. A lexically scoped system would probably execute at only half the speed. Gold Hill is foregoing the complete compatibility of GCLISP with Common LISP until its compiler becomes available. The compiler should take care of the speed issue.

Thorough is the operative word here. The system requires 512K to run. It has all of the features one would expect of a complete LISP system priced at $495. The documentation consists of Patrick Henry Winston and Berthold Klaus Paul Horn's *LISP* (second edition), Guy Steele's COMMON LISP language definition manual, and a user's manual. Besides the LISP interpreter, the system has an excellent GMACS editor and a first-rate, computer-based instruction course called the San Marco LISP Explorer.

The GMACS editor is similar to the EMACS editor and has some very helpful features, including a blinking indication of matching parentheses. Operating the editor is relatively straightforward. The only downside to the editor is its frequent interruption for garbage collection. This can prove annoying on occasion.

Evaluation of LISP forms can be done from the editor. This neutralizes the somewhat irritating necessity of saving the edited text on a disk file and then reloading the file into the LISP interpreter for running. Multiple window buffers can be viewed and edited simultaneously.

The San Marco LISP Explorer is fun. It contains a feature called the LISP Inspector that is nearly undocumented in the manuals. The LISP Inspector is a graphics and text showpiece that visually traces the execution of a LISP program. It is one of the slickest tools I have ever seen. The rest of the Explorer is a multipart tutorial on LISP. It does an excellent job of adding hands-on support and simplifying explanations of topics addressed in the Winston and Horn text.

The LISP interpreter itself is complete and stable. It contains a full complement of list operations, read and splice macros, recursive and iterative control structures, and integer and floating point math. The transcendental math functions have full

```
Factorial benchmark

(DEFUN FACT (LAMBDA (X)
      (COND
        ((LT X 2) 1)
        (T (* X (FACT (SUB1 X))))))

Invoking statement: (FACT 250)
```

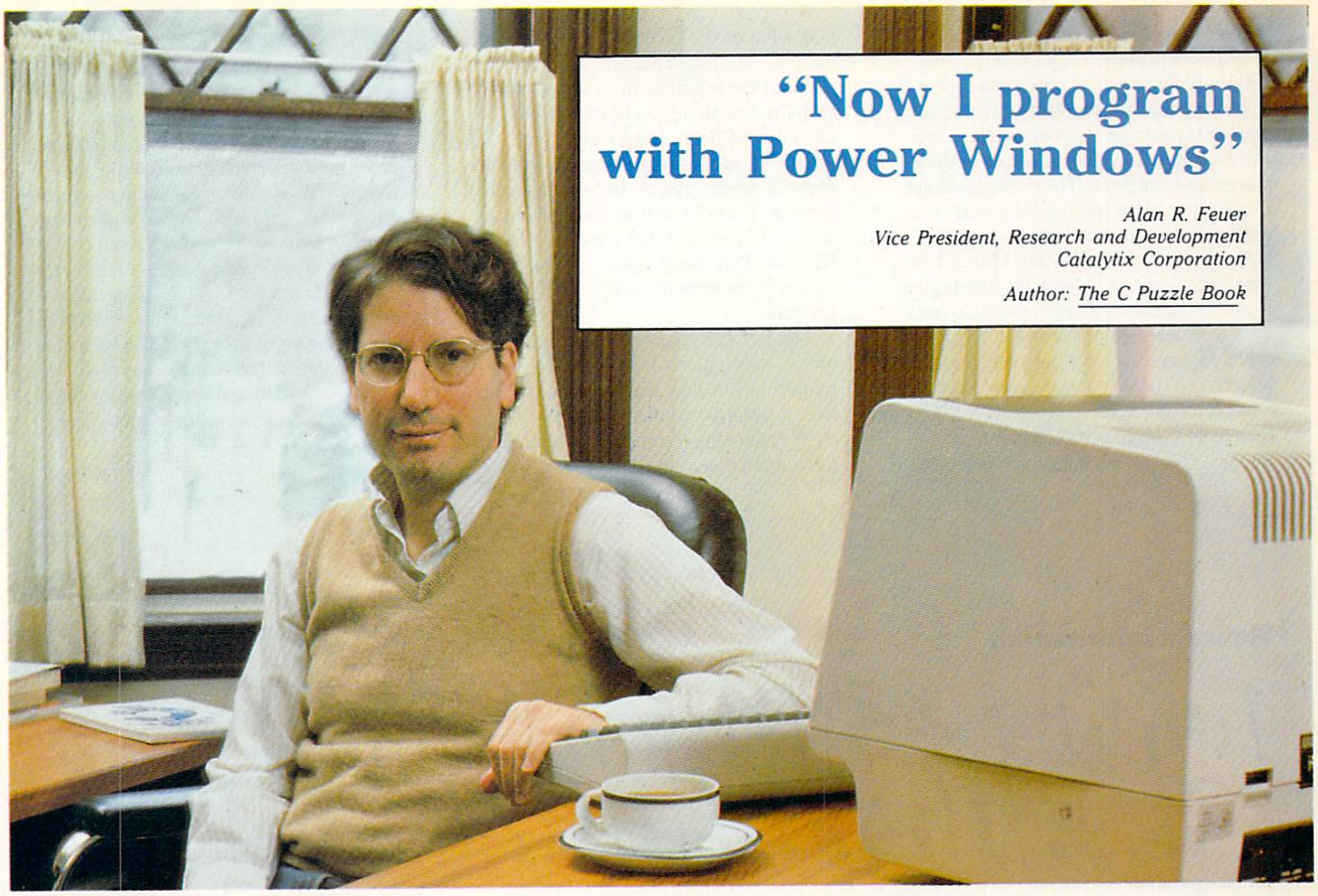
Listing 1.

```
Fibonacci benchmark

(DEFUN FIB (LAMBDA (X)
     (COND
         ((LT X 2) 1)
         (T (+ (FIB (SUB1 X)) (FIB (- X 2)))) ) )

Invoking statement: (FIB 17)
```

Listing 2.

```
MYREV/APP benchmark

(DEFUN MYREV (LAMBDA (X)
  (COND
    ((NULL X) NIL)
    (T (APP (MYREV (CDR X)) (LIST (CAR X)))) ) ))

(DEFUN APP (LAMBDA (X Y)
  (COND
    ((NULL X) Y)
    (T (APP (MYREV (CDR (MYREV X))) (CONS (CAR (MYREV X)) Y))) ) ))

Invoking statement: (APP '(A B C D E F G) '(A B C D E F G))
```

Listing 3.


"Now I program with Power Windows"
Alan R. Feuer
Vice President, Research and Development
Catalytix Corporation
Author: The C Puzzle Book

CCA EMACS...The Most Powerful Editor Environment Available for Unix and VAX/VMS

"Programming with CCA EMACS, I can look at two or more files at once in different windows and then move text between them."

Alan Feuer is just one of many demanding programmers who have discovered that CCA EMACS™ makes program editing and system development much easier and faster. And "power windows" are only part of the reason Alan Feuer uses CCA EMACS. . .

Unprecedented power, speed, functionality, extensibility, pliability, and consistency across systems and on *any* terminal are others. CCA EMACS includes close to 400 built-in commands which let you do any job with only a few keystrokes, even the kinds of things that are difficult or impossible with other editors. And with our full Common Lisp-based extension language, Elisp™, you can customize CCA EMACS to meet all your specific program needs.

CCA EMACS has two extensive recovery facilities to protect against system failures. Supported by a full online documentation package, including tutorial, the system can be used by beginners and experts alike.

This complete kit of editing tools runs under Berkeley Unix™ (4.1BSD and 4.2BSD), Bell Unix (Systems III and V), Xenix™, and VAX/VMS™.
Binary prices range from $380 to $850 for Unix to $1900 for VMS.

CCA Uniworks, Inc.
Productivity Tools for Programmers
20 William Street, Wellesley MA 02181

CIRCLE 60 ON READER SERVICE CARD

For more information or to place an order call our customer representatives at

800-222-0214

in MA (617) 235-2600
or mail this request form today.

Please send me information on:
☐ CCA EMACS ☐ The Safe C Development Tools
☐ AI Development Tools ☐ Your complete line of state-of-the-art programming tools

☐ Please send license forms
Name ____________
Title ____________
Company ____________
Address ____________
City, State, Zip ____________
Phone (   ) ____________

CCA UNIWORKS, INC.
20 William Street Wellesley, MA 02181

A Crowntek Company

Unix, VAX and VMS and Xenix are trademarks of Bell Laboratories, Digital Equipment Corporation, and Microsoft Corporation, respectively. Safe C is a trademark of Catalytix Corporation. CCA EMACS and Elisp are trademarks of CCA Uniworks, Inc. CL785

8087 support (in fact, most of them require the presence of an 8087). Only single-dimensioned arrays are supported.

Among the more advanced features are closures, package support, windowing and I/O streams, and full use of optional and rest parameters. The debug package is quite complete and has the usual complement of trace, break, and single-stepping functions. The San Marco LISP Inspector is also available for debugging.

All in all, this is a very complete package, with features not found in any of the other packages. I have only one real criticism of GCLISP—it is copy protected.

Integral Quality IQLISP

Integral Quality's IQLISP is like a tank. It might not be the fastest thing on the road, but it is solid, dependable, full of features, and gets the job done in a no-nonsense fashion. It also has its fair share of chrome and flash. But unlike a tank, IQLISP is reasonably priced. It is a commercial grade system. In fact, several of the new Texas Instruments AI products for the TI Professional were written with IQLISP. This should give you some indication of the stability and maturity of this product.

The IQLISP package contains an interpreter, debugger, structure editor, and numerous development utilities. At this time it has no compiler, but Integral Quality is promising a compiler sometime in the not too distant future.

The IQLISP debugger and error monitor are the best of all the packages that were tested. The development environment contains a set of functions to facilitate the inspection of the stack after an error has occurred. Additionally, the monitor allows the programmer to restart computation from any of the suspended functions. A nicely laid out stack control window is presented whenever the error monitor is invoked. A lucid English error message appears, along with the expression that caused the error. The user may then use the cursor control keys to climb up and down the stack and modify the offending expressions or values. Very slick.

This system also provides a package

General information

Manufacturer and product	Editor	Numbers	8087	Array dim	Graphics	Package	Debug	Compiler	Window	Assembly interface	Read/splice macros	Function types	I/O support	I/O streams	Mapping	Class system
Levien Instrument BYSO LISP	FS	int	no	multi	no	no	yes	no	no	no	no	FE	S	no	yes	no
ExperTelligence Inc. ExperLISP	FS	int,real	—[1]	multi	yes	yes	yes	yes	yes	yes	yes	FE	RS	yes	yes	yes
Gold Hill GCLISP	FS	int,real	yes	single	yes	yes	yes	no	yes	no	yes	FLSR	RS	yes	yes	yes
Integral Quality IQLISP	struc	int,real	yes	multi	yes	yes	yes	no	yes	yes	yes	FLSR	RS	yes	yes	no
Software Toolworks LISP/80																
Norell Data Systems LISP/88	struc	int	no	no	no	no	yes	no	no	yes	no	FE	S	no	yes	no
Microsoft MuLISP	FS	inf/int	no	no	yes	no	yes	pseudo	no	yes	yes	FLSR	RS	yes	yes	no
The Lisp Co. TLC-LISP	FS	int,real	yes	single	yes	yes	yes	yes	yes	no	yes	FLSR	RS	yes	yes	yes
Northwest Computer Algorithms UO-LISP	FS	inf/int	no	single	yes	yes	yes	yes	no	yes	yes	FLSR	RS	yes	yes	in source
ProCode International Waltz LISP	FS	inf/int	no	single	no	yes	yes	no	no	yes	yes	FLSR	RS+	yes	yes	no
XLISP	no	int	no	multi	yes	yes	yes	no	no	C source	yes	FE	S	no	yes	yes

1. See IEEE standard.

FS = Full screen.
struc = Structure.
inf/int = Infinite precision integers.
FE = FEXPR EXPR.
FLSR = FEXPR LEXPR subroutine.
S = Sequential.
RS = Random/sequential.
RS+ = Random/sequential with additional types to find.

Table 2.

system, pretty printer, windows, graphics, floating point math with 8087 support, multidimensional arrays, read and splice macros, and a good assembly language interface. File control is well implemented, as is a first-rate DOS interface.

Suspended environments can be saved and restored with IQLISP. Garbage collection is relatively fast and unobtrusive. Oblist control is extensive. The DEVELOP.SYS package contains packaging commands and a full complement of structured programming functions. Using the structured commands can help ease the transition from a standard algebraic-type language to LISP. It is a big plus for this package.

Documentation for IQLISP generally is good, although the manual could do with more examples. Overall, this is one fine package.

Software Toolworks LISP/80 Norell Data Systems LISP/88

Software Toolworks' LISP/80 and Norell Data Systems' LISP/88 seem to be the same package in their MS-DOS incarnations. What is said here applies to both interpreters. LISP/80 is friendly and slow compared to other implementations. The price and features make this an ideal LISP for someone who wants to learn the language.

Walt Bilofsky, the author of LISP/80, says that some of his customers find it quite adequate for serious applications as well, but he does not make claims nor try to sell it for that sort of use. LISP/80 and LISP/88 are patterned after the INTERLISP dialect of LISP and are integer-only systems.

For LISPs in this price range ($39.95), the packages are feature-rich. In addition to the interpreter, LISP/80 and LISP/88 contain a structure editor, pretty printer, several example programs (including a version of the ELIZA program), and a fairly complete manual.

The manuals are divided into two parts: a humorously written tutorial and a language reference. The useful trace/break-type debuggers feature a backtrace function. LISP/80 and LISP/88 are small LISPs and spend a lot of time doing garbage collections. Use of available space can be optimized by reserving list cells and stack space when invoking the interpreter.

Assembly language subroutines can also be written and interfaced. LISP/80 and LISP/88 are written in C/80 and use that language's subroutine calling conventions. This procedure is quite straightforward and is well documented in the manual.

Microsoft MuLISP

The Microsoft marketing boys are at it again. The front cover of the MuLISP binder calls it an artificial intelligence development system. This almost leads one to suspect that next they'll be bundling an editor and macro assembler together to sell as an integrated spreadsheet development system.

MuLISP was one of the first microcomputer implementations of LISP. It was developed by Al Rich, David Stoutmeyer, and Roy Feldman of the Soft Warehouse. What they designed was a high-performance, general purpose LISP programming system that is very good and very fast.

The MuLISP package is also very complete. It contains an interpreter, a super editor called MuSTAR, a complete computer-based tutorial, and a competent debugger. In addition, it contains a number of compatibility packages that allow it to function like INTERLISP and MACLISP. The package is rounded out by a number of good demonstration programs.

MuLISP is quite compact. The MS-DOS version can run (albeit not too efficiently) in as little as 64K. The MuLISP system itself is a combination pseudocode compiler and interpreter. It is very fast. The speed, however, comes at a cost. Error trapping is kept at a minimum. The pointer system is a closed pointer universe, which prevents the implementation of closures. But these are minor problems and do not impact the overall quality of the package.

MuLISP has a full complement of the things that define a good LISP package: sequential and random access I/O, I/O streams, highly efficient string functions, infinite precision integer arithmetic (no floating point support, however), read and splice macros, and a table-driven scanner. A graphics support package is also included. MuLISP does not support arrays or a package system.

The MuSTAR editor features a console customization function for adaptation to almost any computer. The editor allows the editing of functions, variables, and property lists. Evaluation of the LISP form from within the editor also is provided.

The Lisp Company TLC-LISP

TLC-LISP, put out by The Lisp Company, is one of the more complete packages in this group and features one of the nicest documentation packages. It isn't often that a manual is written with such style and elegance that it can be read just for reading's sake. And the LISP tutorial portion of the manual compares favorably with many of the LISP texts I have read in the past.

TLC-LISP is sleek and elegant. The interpreter and compiler fulfill the promise of the manual. This package is one of a handful of LISPs that include a full class system, and the class system implemented here is very LISP-machine-like in nature.

C

HOW EASY DEVELOPING PROGRAMS CAN BE WITH THE C UTILITY LIBRARY!

COMPLETE. Over 300 tested and well documented functions. All features of the PC &AT are at your finger tips.

ADVANCED. Features like windows, data entry fields, switching displays, and batch file execution are supported.

GRAPHICS. A complete set of fast business graphics functions. Our low level graphics routines are the fastest in the business.

SOURCE. All source code is included. 95% of the library is written in C. Only functions demanding fast execution speed are in Assembler.

COMPATIBLE. With Lattice, Microsoft(3.0), CI-C86, Aztec, DeSmet, Wizard, and Mark Williams. All memory models are supported.

MUCH MORE. Dozens of string functions, the best time & date math and formatting, DOS directory and file mgmt., keyboard control, polled async communications, and more. **NO ROYALTIES.** We **SUPPORT** what we sell.

C UTILITY LIBRARY - $185
COMPILERS: Lattice $349, CI-C86 $329, Mark Williams $449. **Save $40-$50** with compiler & library package.

Specify compiler and version number when ordering. Add $4 for UPS or $7 for UPS 2-day shipping. NJ residents add 6% sales tax. VISA, MC, Chks & qualified PO's.

ESI
Essential Software, Inc.
P.O. Box 1003
Maplewood, New Jersey 07040
914/762-6605

CIRCLE 33 ON READER SERVICE CARD

C

SOFTWARE DEVELOPERS!

V - FILE
THE VIRTUAL MEMORY FILE MANAGER

Let V-FILE save precious development time & cost as you create efficient applications with the power of VIRTUAL MEMORY.

DON'T RE-INVENT THE WHEEL

Why spend weeks or months coding and debugging file and memory management systems when you can order V-FILE today. V-FILE is a library that you can link with your code to provide sophisticated virtual file and memory management — allowing you to concentrate on developing your application.

VIRTUAL DATA OBJECTS SUPPORTED!

Data is referenced by using VIRTUAL MEMORY DATA HANDLES. Your code doesn't need to know whether the data is actually on disk or in RAM. Swapping between disk and RAM and updating files on disk is handled automatically and transparently! Complex VIRTUAL DATA STRUCTURES can be created by linking with data handles instead of pointers.

CHECK THESE FEATURES!

- Multiple, independent swap buffers
- Multiple files per swap buffer
- Highly efficient swap algorithm
- Automatic file updating
- Data prefetching supported
- Data may be locked in memory
- Memory buffers may be flushed
- Makes full use of extended memory on IBM PC/AT
- SOURCE CODE AVAILABLE
- NO ROYALTIES REQUIRED

Supports Dos 2.00+ with Lattice & Microsoft C compilers
Supports Microsoft windows

$299

V·FILE TM

Contact:
MindBank, Inc.
4620 Henry Street
Pittsburgh, PA 15213
412/683-9800

VISA/MASTER CARD ACCEPTED

CIRCLE 63 ON READER SERVICE CARD

It supports separate name spaces in a monolithic address space. The author says this is in anticipation of very large address spaces (and therefore very large projects) that will be available in the near future. This technique will be of substantial use as the size and number of embedded LISP applications expand. TLC-LISP contains the full complement of class operators *CLASS, INSTANCE,* and *INHERITANCE.*

Like IQLISP, TLC-LISP includes many structured forms of iterations: *CASE* and *CATCH-THROW* pairs, and *DO*. The addition of these explicit control structures to the implicit control (call-by-value and recursion) of LISP provides a very powerful set of control tools.

TLC-LISP's interpreter is quite slow, but the compiler increases efficiency fourfold or better. For instance, the MYREV/APP benchmark required nearly 4 min to finish running interpretively. But after compiling, the system executed in just over 50 sec. The debug package allows tracing compiled functions.

There is debate as to whether floating point math is necessary for a symbolic language. I, for one, use LISP for the development of vision and imaging systems. Convolution would be impossible without it. TLC-LISP supports floating point math and the 8087 chip.

Other features worth noting are full graphics, turtle graphics, a DOS interface, an autoloader, I/O streams, a full screen editor, and a library of IBM PC specific features. Overall, this was one of the best packages tested and is highly recommended.

Northwest Computer Algorithms UO-LISP

Northwest Computer Algorithms' UO-LISP was a surprise. After seeing the ads for the package, I was anticipating a stripped-down learning system similar to LISP/80. Wrong. The UO-LISP development system is full and complete. It is also unbelievably fast.

The compiler version for MS-DOS is an extended version of UO-LISP 1.16a (the CP/M version). The interpreter is relatively slow. Once compiled, however, UO-LISP turned in the fastest speeds of any of the LISPs tested. The compiler, incidentally, can be toggled to produce a complete assembly code listing that can then be further optimized by the programmer.

The package is feature-rich and worked in a predictable and reliable fashion. The only glitch occurred when loading a buggy program from a disk. This caused the evaluator to stick in a loop that neither Control-C nor Control-ALT-delete could terminate. The machine had to be switched off and then back on to regain control.

Calling UO-LISP complete may be a bit of an understatement. Besides the interpreter and compiler, UO-LISP contains a complete debugging (trace) package and an execution profiler. Utility packages include three editors—structure, character, and full screen—an RLISP dialect compatibility package, the LISPTEX text formatter, a BIGNUM and FIXNUM system for handling large integers and fixed point numbers, sort/merge, two printing systems (pretty printer and terse printer), and one of the better screen and graphics driver packages I've seen. One of my favorite features was a history-saving read loop that allowed me to play back a LISP session. PC-DOS interface routines also are included.

Documentation is copious. The manual is nearly 400 pages long. It is clearly written and contains hundreds of examples. At $150, this is certainly a bargain.

ProCode International Waltz LISP

Waltz LISP, by ProCode International, has been a long-time favorite of the CP/M crowd and deservedly so. It occupies less than 30K of RAM. It provides an exceptionally efficient and friendly working environment by virtue of the fact that the debugger and error-handling routines remain on-line (in RAM) at all times rather than relying on overlays.

It has the fastest and fanciest file access system of the LISPs tested (this is where the benchmarks can lead one astray—if any of the benchmarks were to test random access file handling, Waltz LISP would have left the others in the dust). The elegance of the file handler allows for the near instantaneous access of any byte in the file. This enables easy programming of virtual function storage and, to a great extent, entire virtual programs. Waltz LISP does not limit the user to the number of open files dictated by the *FILES = NN* declaration in the CONFIG.SYS file. Waltz creates its own handles and provides for open file manipulation up to the available memory limit. Waltz LISP fully supports the MS-DOS 2.x file system, including the subdirectory structure.

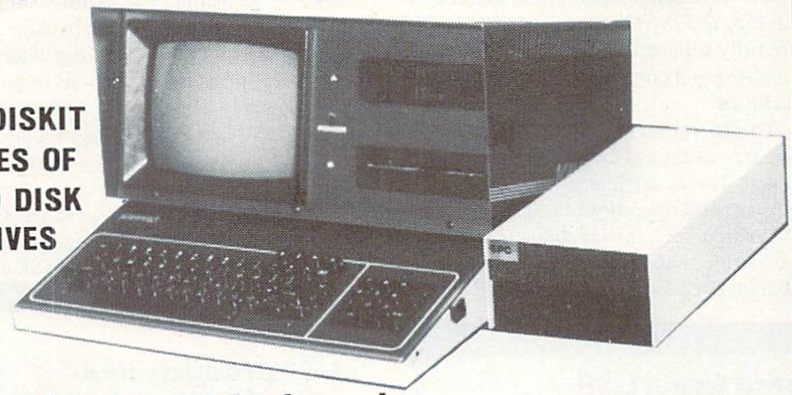
The Waltz LISP package contains a WordStar-like full screen editor that allows editing files of almost any size from within the LISP environment. In addition, ProCode plans to add another RAM resident editor in the near future.

As with most of the better LISPs, Waltz LISP includes the standard suite of structured programming control functions (*do, let, catch,* etc.). Large integers with selectable radix are supported up to 611 digits. Floating points are not supported. The usual complement of mappers, pretty printers, I/O streams, and sort/merge are included.

The documentation for this product,

FOR THE SERIOUS KAYPRO® USER

THE DISKIT SERIES OF HARD DISK DRIVES



now with

ZCPR3 by Echelon, Inc.

Now you can add from 5 to 40 Megabytes of fast-access Winchester storage to your KAYPRO 2, 4, or 10. The DISKIT is only 4 inches high; 5.7 if you get the two drive model with the *removable* 5 or 10 Mb. cartridge, and weighs less than 10 pounds. Easily disconnect DISKIT from the computer whenever you want, and if more capacity is required, just swap your drive for a larger model.

Our DISKIT Model 10 has 10.8 Megabytes of *formatted* capacity . . . **20%** more than a Kaypro 10, and runs about twice as fast. Installs in minutes.

Call SPC now and ask for more information. Quantity and prepayment discounts are available.

SYSTEMS PERIPHERALS CONSULTANTS
9747 Business Park Avenue
San Diego, CA 92131
(619) 693-8611

CIRCLE 38 ON READER SERVICE CARD

THE WORLD'S FASTEST MOST POWERFUL 8080 RELOCATING MACRO ASSEMBLER

SLRMAC™

ONLY **$49.95**

AT THIS PRICE, SHOULD YOU BE WASTING YOUR TIME USING SOMETHING ELSE?

This is what they said about Z80ASM, our Z80 assembler. Now the same features and performance are available in our Intel Mnemonic product. SLRMAC is compatible with M80 in .8080 mode, with many extensions. Too many features to list here.

To order or to find out more about our complete family of development tools, call or write:

SLR Systems

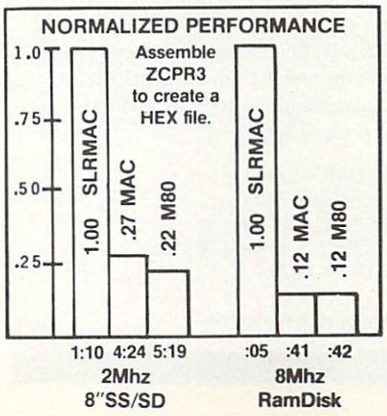
1622 N. Main St., Butler, PA 16001
(800) 833-3061, (412) 282-0864
Telex 559215 SLR SYS

C.O.D., Check or Money Order Accepted.

Add $3 shipping.
Z80 CP/M compatibility required.

". . a breath of fresh air . ."
Computer Language, Feb. 85

". . in two words, I'd say speed & flexibility",
Edward Joyce, Nov. 84
Microcomputing



CIRCLE 28 ON READER SERVICE CARD

YOU NEED A GOOD C LIBRARY

C POWER PACKS™

COMPLETE SOURCES NO ROYALTIES

COMPREHENSIVE C Power Packs include over 1000 functions which provide an integrated environment for developing your applications efficiently. "This is a beautifully documented, incredibly comprehensive set of C Function Libraries."
– Dr. Dobb's Journal, July 1984

USEFUL "...can be used as an excellent learning tool for beginning C Programmers..."
– PC User's Group of Colorado, Jan. 1985

FLEXIBLE Most Compilers and all Memory Models supported.

RECOMMENDED "I have no hesitation in recommending it to any programmer interested in producing more applications code, using more of the PC capabilities, in much less time." — Microsystems, Oct. 1984

- **PACK 1: Building Blocks I** $149
  DOS, Keyboard, File, Printer, Video, Async
- **PACK 2: Database** $399
  B-Tree, Virtual Memory, Lists, Variable Records
- **PACK 3: Communications** $149
  Smartmodem™, Xon/Xoff, X-Modem, Modem-7
- **PACK 4: Building Blocks II** $149
  Dates, Textwindows, Menus, Data Compression, Graphics
- **PACK 5: Mathematics I** $99
  Log, Trig, Random, Std Deviation
- **PACK 6: Utilities I** $99
  (EXE files)
  Arc, Diff, Replace, Scan, Wipe

Master Card/Visa, $7 Shipping, Mass. Sales Tax 5%

ASK FOR FREE DEMO DISKETTE

NOVUM ORGANUM INC.
SOFTWARE HORIZONS INC.

165 Bedford St., Burlington, MA 01803
(617) 273-4711

CIRCLE 25 ON READER SERVICE CARD

over 300 pages, is quite good. It is written with a sensitivity for the beginner but is not short on the nuts and bolts information required by the experienced LISP practitioner.

XLISP

XLISP is a public domain LISP interpreter. David Betz, the author, provides both an executable file and the source for the interpreter in C. The documentation is sketchy but provides the necessary syntactical information for the experienced LISP programmer.

What is surprising about XLISP is its completeness. It incorporates a full object-oriented programming system. *CLASS, INSTANCE,* and *INHERITANCE* are fully supported. The usual list-handling and control forms also are included.

The operation of XLISP was entirely bug free as far as I could tell. The version tested, 1.4, is sufficiently mature to ensure good operation.

Written in C, the language is easily extensible. Numerous bulletin boards contain code for the addition of floating points, graphics, and communications to the package. I have talked to the author on several occasions, and he fully supports the idea of additions to the package, but he requests that those making changes forward a copy of the extensions to him. He is trying to work as a clearinghouse for the XLISP language and appreciates knowing what's going on with the language.

For those who want to get their feet wet with symbolic programming, I can think of no better way than with XLISP and a good introductory text to LISP.

For the Macintosh: ExperTelligence ExperLISP

ExperLISP is a result of ExperTelligence Inc.'s historical support for artificial intelligence. The company claims it is the first compiled implementation but Table 2 shows otherwise. ExperTelligence plans a future release that will correct some of the bugs reported to date.

While the Mac does not have color graphics at this time, you are able to specify five shades between black and white. The sample graphics program offers two-dimensional, three-dimensional, and spherical bunny graphics like Logo's. A complex figure was drawn in three windows simultaneously. It was quite impressive.

Programming errors are indicated and the next release promises to pinpoint the variables within the specified line even better. A listener window keeps track of all interactive code and the second window holds the program in an edit buffer. It is very easy to click back and forth between the two windows to compile your code, either selected sections or the entire buffer. Compiled code is not saved in the initial release. A file compiler will be available in the near future, as well as the ability to save a snapshot of memory. ExperLISP is lexically scoped, so symbols are bound within the scope of their containing body.

The benchmark times are impressive and the number of functions and commands (403 in version 1.0) extensive. The cost is $1.23 per function. If this is within your budget, then I highly recommend this product. It has the added advantage of calling assembly language routines directly. Numbers may be integer or real (floating point), and 80-bit precision is offered with the Apple SANE package, IEEE standard. Garbage collection uses the mark and sweep method. A future release will be able to control this space.

Also provided with this product is *An ExperLISP Reference Guide*, 189 pages; David S. Touretzky's *LISP: A Gentle Introduction to Symbolic Computation*, 384 pages; two disks; and four pages of notes to ensure compatibility between ExperLISP and Touretzky's LISP. You can have it either way.

ExperLISP has three sample programs: search, a case-sensitive search; graphics; and Quicksort, a numerical sort. Because LISP has several dialects, the guide with each function indicates whether it is primitive, what type it is, the version it will be implemented in (1.0 or 1.1), and if it came from Common LISP, ZetaLISP, or the Macintosh Toolbox (ROM) or is unique to ExperLISP.

As ExperLISP is under active development, those who support the release of 1.0 will receive updates to release 2.0 at a nominal cost (disk plus shipping). The current release makes extensive internal use of the class system. Later you will be able to define your own classes and methods. See Patrick Henry Winston and Berthold Klaus Paul Horn's *LISP* (second edition) for object-oriented programming. Apple is introducing Object Pascal, which will include classes and objects to handle the user interface portions of a Macintosh application. This will be called MacApp. A version will be in release 2.0 of MacApp in LISP. See also K.J. Schmucker's *Object-Oriented Programming on Macintosh and Lisa.*

Overall, this ExperLISP package is excellent, and the future offers many new advancements to drive LISP to even greater heights.

ExperLISP—$495
ExperTelligence Inc.
559 San Ysidro Rd.
Santa Barbara, Calif. 93108
(805) 969-7874

By Robert Ashworth

LISP products and manufacturers

BYSO LISP—$125
Levien Instrument
Box 31
McDowell, Va. 24458
(703) 396-3345

GCLISP—$495
Gold Hill Computers
163 Harvard St.
Cambridge, Mass. 02139
(617) 492-2071

IQ-LISP—$175
Integral Quality Inc.
P.O. Box 31970
Seattle, Wash.
(206) 527-2918

LISP/80—$39.95
Software Toolworks
15233 Ventura Blvd., Ste. 1118
Sherman Oaks, Calif. 91403
(818) 986-4885

LISP/88—$49.95
Norell Data Systems
P.O. Box 70127
3400 Wilshire Blvd.
Los Angeles, Calif. 90010
(213) 748-5978

MuLISP—$300 (IBM) $250 (CP/M)
Microsoft
10700 Northup Way
Box 97200
Bellevue, Wash. 98009
(800) 426-9400

TLC-LISP—$250
The Lisp Co.
P.O. Box 487
Redwood Estates, Calif. 95044
(408) 354-3668

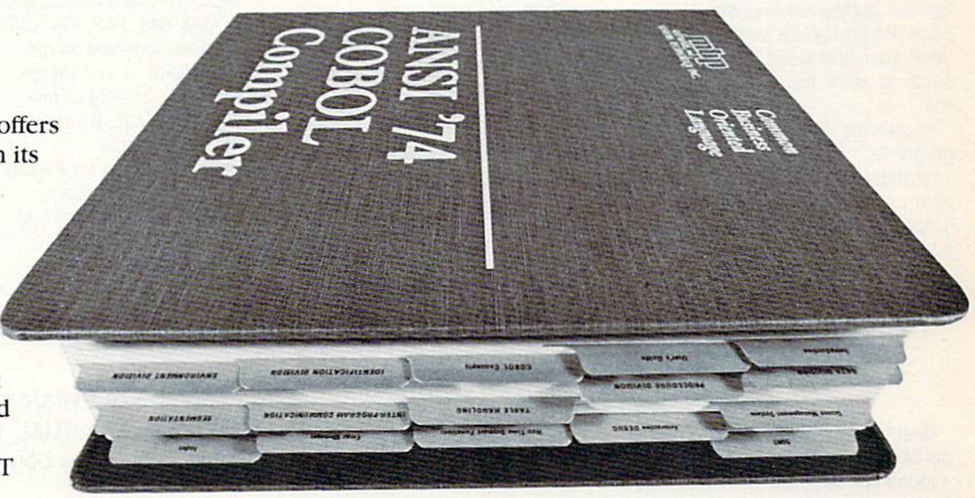
UO-LISP—$150
Northwest Computer Algorithms
P.O. Box 90995
Long Beach, Calif. 90809
(213) 426-1893

Waltz LISP—$169 (IBM and CP/M)
ProCode International
15930 SW Colony Pl.
Portland, Ore. 97224
(503) 684-3000

XLISP—$7 (add $3 for overseas)
PC/BLUE Users Group
The New York Amateur Computer Club Inc.
Box 106
Church St. Station
New York, N.Y. 10008

mbp COBOL
for your IBM/PC

The new standard for convenience.



Now, the mbp COBOL Compiler offers unrivaled convenience to go with its unmatched performance.

Here are the convenience features you've wished for:

1) an enhanced Screen Management System with program-controlled video attributes and color; **2)** support for PATH & sub-directories; **3)** DOS command execution from within a COBOL program; **4)** 'permanent' DEFAULT modification.

The new mbp Compiler has them all! And they're exclusives: you get them *only* with mbp.

Plus, it's 4 times faster.

Because the mbp COBOL Compiler generates native machine language object code, it executes programs *at least* 4 times faster (see chart). Now, we've made that performance even more convenient to use.

GIBSON MIX Benchmark Results
Calculated S-Profile
(Representative COBOL statement mix)
Execution time ratio

mbp* COBOL	Level II* COBOL	R-M* COBOL	Microsoft* COBOL
1.00	4.08	5.98	6.18

*128K system with hard disk required. IBM PC & AT are IBM TMs; Netware is a Novell, Inc. TM; Level II is a Micro Focus TM; R-M is a Ryan-McFarland TM; Microsoft is a Microsoft TM.

The complete COBOL.

An Interactive Symbolic Debug Package included standard; Multi-keyed ISAM structure; SORT & CHAIN; GSA certification to ANSI '74 Level II; IBM/PC-AT and TI Professional compatibility; with mbp, you get it all. Optional: Novell NetWare interface.

mbp COBOL: the choice of professionals.

It's no surprise more and more companies like Bechtel, Bank of America, Chase, Citicorp, Connecticut Mutual, Hughes Aircraft, McDonnell-Douglass, and Price-Waterhouse choose mbp COBOL.

Make it your choice, too. Just send the coupon, or call, for complete information. Today.

mbp COBOL. $1000

Please send complete mbp COBOL information to:

NAME ______________________

COMPANY ______________________

ADDRESS ______________________

CITY/STATE/ZIP ______________________

PHONE ______________________

mbp Software & Systems Technology, Inc.
7700 Edgewater Drive, Suite 360
Oakland, CA 94621
Phone 415/632-1555

mbp®

CIRCLE 53 ON READER SERVICE CARD

C Programmers:

Consider 104 Ways To Be More Productive

If you find and choose the right development software, you can: cut development effort, make impractical projects feasible, and eliminate unproductive, frustrating aspects of programming.

Confused? We'll help you sort thru the huge number of alternatives. Call for comparisons or information.

We carry 27 C Compilers, 4 C Intepreters, 49 Support Libraries, 5 C source debuggers, and 19 other C Add-ons for programming with MSDOS, Macintosh, or CP/M — more than 104 products, really. Here are some of the best products available:

Learn C Programming Only $95

"Introducing C" Interpreter

Computer Innovations has done it again! This interactive implementation is combined with a full screen editor and a thorough, self-paced manual.

You can develop programs faster by getting immediate feedback. Programs will start instantly upon your command. There is no need to wait for "compile and link."

Introducing C includes demo programs, powerful C language interpreter, complete C function library, full screen editor, color graphics, and C language compatibility. PCDOS $95

Simplify Screen Management

Windows for C

Keep your software up to date with the latest screen management features:

- Pop up menus and help files
- Instant screen changes
- Multiple windows
- Complete color control

Windows for C offers all of these plus much more in an integrated, compact, easy-to-use library of object code functions. Thorough, reference manual. Support for all memory models of popular C compilers. New version 3.1 offers enhanced portability and TopView compatibility. Full source available. MSDOS $180

SORT/MERGE Files for Clean, Fast Maintenance

with OPT-TECH SORT

Performance should not suffer with DOS or other "free" sorts. ISAMs alone are slow when 10% or even less is changed/added.

OPT-TECH includes:

- CALLable and Standalone use
- C, ASM, BAS, PAS, FTN, COBOL
- Variable and fixed length
- 1 to 9 fields to sort/merge
- Autoselect of RAM or disk
- Options: dBASE, BTrieve files
- 1 to 10 files Input
- No software max for # Records
- All common field types
- By pass headers, limit sort
- Inplace sort option
- Output = Record or keys

Try what you're using on an XT: 1,000 128 byte records, 10 byte key in 33 seconds. MSDOS $90

Add Communications Features to Your Programs

Greenleaf Comm Library

Greenleaf now enables you to communicate with remote systems or databases with an asynchronous communications library for C.

Individual transmission and reception ring buffers combine with an interrupt driven system. This eliminates the extra function of separately calling up the communications program.

Included are 1 library/object files, 100 functions; 100 page manual, complete source code, library tailor-made to suit compiler and memory, Hayes-compatible modem commands, and a complete sample file transfer program. MSDOS $169

Which Compiler Features Do You Need?

Optimizing C86 Compiler

Over the years the Optimizing C86 has evolved to be the most complete set of C compiler tools. It includes utilities, a rich library, and thorough tech support. In line 8087/287 routines run up to 100 times faster than the 8086 math package. The source code to all routines is included, so you have complete control over how they work. Thorough ROM support, Intel UDI & VMS cross versions are available.

More of the features you want include:

- special IBM-PC library • 2 math and 2 I/O libraries
- full memory utilization of the 8086/88/186/286
- compatibility with most commercial libraries
- object and source module librarian MSDOS $339

Get File Access with TIGHTER Control

db_VISTA Data Management

Full source, no royalties and "normal" indexed file management are part of db_VISTA. Get more for the price of only an ISAM.

You can minimize data stored and access records even faster and more logically than just using indexes. Example: address and transaction data should not require redundant storage of customer names or numbers. Use pointers. Related data fields point to other related groups — the "network model" of data.

Use db_VISTA as a "normal ISAM" or save programming time, access time and file size. Lattice, C86, Williams, Desmet, Microsoft C. MSDOS. Source $450, Object $275.

Inventive Programming Becomes Possible with 300+ ESSENTIAL, tested, fast, routines to Rely On.

C Utilities Library by Essential Software

Recent Enhancements to Graphics, Windows, AT Support

Every application you write is likely to require functions where you feel like you are "reinventing". Don't. Even if you use only 5% of this library, you will come out ahead on schedule and cost.

Full business Graphics, Window support, polled Communications, and Data Entry support have recently been added/upgraded along with more functions for DOS Interface and AT support. String handling, screen control, "word processor" functions, memory management, directory and path access, date handling, program chaining, keyboard and printer control are traditional strengths.

Full source code is included. No royalties are charged to include functions in your programs. 95% are C for portability and to make it practical for you to understand or modify them.

Lattice, Microsoft, C86, Mark Williams, Aztec, Desmet and Wizard C are supported. Specify which you need.

Substantial time, effort, testing and attention has been invested by Essential Software developing, documenting and supporting this comprehensive library. Make new projects practical and interesting. Use this tested and reliable library.

Some functions are PC-specific. Most support any MSDOS. $159.

Call for details, comparisons, or for our "C Extras Packet" with over 50 pages of information about C support products.

THE PROGRAMMER'S SHOP

The programmer's complete source for software, services and answers

128-LC Rockland Street, Hanover, MA 02339 (617) 826-7531 (800) 421-8006

Ask about COD and PO's. All formats available. Prices subject to change. Names of products and companies are generally their trademarks.

CIRCLE 117 ON READER SERVICE CARD

PC PROLOGs

By Namir Clement Shammas

PROLOG is one of the latest artificial intelligence languages to gain popularity. In this review we will look at the various PROLOG packages that currently are available for the IBM PC microcomputer running under MS-DOS 2.0 or later.

The primary published reference for PROLOG is *Programming in PROLOG* by W.F. Clocksin and C.S. Mellish, published by Springer-Verlag. This book defines a core PROLOG. An alternate PROLOG syntax is found in Micro-PROLOG, which is implemented by LPA Micro-PROLOG.

In this review we will look at each package's extension and how it reflects the authors' creativity. Several of the PROLOG implementations reviewed are still undergoing changes and enhancements. These numerous language extensions appear to have created a portability problem for such systems as the IBM PC.

Four benchmark tests were used in this comparison:

- The Sieve of Eratosthenes
- List reversal
- Quicksort
- The Tower of Hanoi.

The Sieve test returns a list of primes up to 100. This does not correspond to the Sieve version used in benchmark tests of other languages such as BASIC, C, Pascal, and Modula-2. The main limitation of this test is the stack overflow due to extensive recursion, which is something PROLOG seems to rely on heavily.

The list reversal was a simple test suggested by Automata Design Associates. Quicksort and the Tower of Hanoi were included to check internal speed in more elaborate schemes. The Tower of Hanoi reflects the combined speed of internal processing and screen display.

While there were other, more elaborate benchmark tests suggested, they used commands peculiar to some implementations. However, since some of the PROLOG implementations are still in their infancy, expect additional development in this area, particularly more predicates and greater speed. So note that the speed benchmarks presented here are not the final word!

Table 1 is a comparison table. Table 2 shows the results of the benchmark tests.

Arity/PROLOG

This heavyweight product comes from Arity Corp., a subsidiary of Lotus Development Corp. Two versions are available: the PROLOG interpreter and the complete system, which includes a PROLOG compiler. The implementation revolves around the Clocksin and Mellish definition, with a few modifications and several versatile additions.

The Arity/PROLOG interpreter displays the ?- question prompt which makes it more accessible to answering questions. To assert any fact from the keyboard, the *asserta()* and *assertz()* predicates must be used. The *assert()* predicate is not implemented.

Arity/PROLOG supports all the PROLOG data types, including floating point numbers. Trigonometric and transcendental mathematical functions are offered, as are the *AND, OR, NOT,* and shift left and shift right bitwise operations.

In addition, Arity/PROLOG supports strings and data base reference numbers. The latter are eight-hexadecimal-digit codes with the tilde symbol to their left, acting as reference points for the data base. As with other PROLOG implementations, Arity/PROLOG supports data classification predicates, including predicates to indicate if a datum is a variable, atom, or integer.

Arity/PROLOG has added predicates to test for floats, strings, and data base reference numbers. In addition, the general purpose predicate *type(Code,Arg)* can be implemented to identify integers, strings, floating point reals, and data base reference numbers.

Arity/PROLOG also provides a number of predicates that perform such string

Benchmark results (sec)

Manufacturer and product	Sieve	Reverse list	Quicksort	Tower of Hanoi
Arity Corp. Arity/PROLOG 3.1	16	6	17	6
Automata Design Associates ADA PROLOG 1.33	46	15	63	9
Chalcedony Software PROLOG V 1.10	SO,CT	CT	CT	CT
Expert Systems International PROLOG-1 2.2	16	4	15	29
Solution Systems PROLOG-86 2.00	8	3	6	8
Logic Programming Associates LPA Micro-PROLOG 3.1	IE	6	SO	21
Logicware MPROLOG 1.5	30	8	15	22

SO = Stack overflow.
IE = Interpreter logical error.
CT = Compulsory tracing.

Table 1.

General information

Manufacturer and product	String manipulation	Floating point	8087 support	Math functions	Modules	Virtual memory	Garbage collection	Random access files
Arity Corp. Arity/PROLOG 3.1	yes	yes	yes	yes	yes	yes	yes	yes
Automata Design Associates ADA PROLOG 1.33	yes	yes	yes[2]	yes	yes	yes	yes	yes
Chalcedony Software PROLOG V 1.10[1]	no	no	no	no	no	no	yes	no
Expert Systems International PROLOG-1 2.2	no	yes	no	no	no	no	yes	yes
Expert Systems International PROLOG-2	yes	yes	yes	no	yes	yes	yes	yes
Solution Systems PROLOG-86 2.00	no	yes	yes	yes	no	no	yes	no
Logic Programming Associates LPA Micro-PROLOG 3.1	yes	yes	no	no	yes	no	yes	yes
Logicware MPROLOG 1.5	yes	no	no	no	yes	no	yes	yes

Manufacturer and product	Screen cursor control	Windows	Date and time	Error handling	Interface with other languages	Compiler + Interpreter	Access DOS	Built-in editor
Arity Corp. Arity/PROLOG 3.1	yes	no	yes	yes	yes	yes	yes	no
Automata Design Associates ADA PROLOG 1.33	no	no	no	no	yes	no	yes	no
Chalcedony Software PROLOG V 1.10	no	no	no	no	no	no	no	no
Expert Systems International PROLOG-1 2.2	no	no	no	yes	yes	no	no	yes
Expert Systems International PROLOG-2	yes	yes	yes	yes	yes	no	yes	yes
Solution Systems PROLOG-86 2.00	yes	no	no	yes	no	no	yes	no
Logic Programming Associates LPA Micro-PROLOG 3.1	no	no	yes	yes	yes	no	no	yes
Logicware MPROLOG 1.5	no	no	no	yes	no	no	no	yes

1. PROLOG 5-plus, a large memory model PROLOG, will be released in September.
2. Special version.

Table 2.

WIZARD C

Fast compiles, fast code and great diagnostics make Wizard C unbeatable on MSDOS. Discover the powers of Wizard C:

- ALL UNIX SYSTEM III LANGUAGE FEATURES.
- UP TO A MEGABYTE OF CODE OR DATA.
- SUPPORT FOR 8087 AND 80186.
- FULL LIBRARY SOURCE CODE, OVER 200 FUNCTIONS.
- CROSS-FILE CHECKS OF PARAMETER PASSING.
- USES MSDOS LINK OR PLINK-86.
- CAN CALL OR BE CALLED BY PASCAL ROUTINES.
- IN-LINE ASSEMBLY LANGUAGE.
- 240 PAGE MANUAL WITH INDEX.
- NO LICENSE FEE FOR COMPILED PROGRAMS.

The new standard for C Compilers on MSDOS!

Only $450

WSS

For more information call (617) 641-2379
Wizard Systems Software, Inc.
11 Willow Ct., Arlington, MA 02174
Visa/Mastercard accepted

CIRCLE 86 ON READER SERVICE CARD

Now With Windowing!
$49.95 Basic Compiler

MTBASIC

Features:

Multitasking	Windowing
Handles interrupts	Interactive
Fast native code	Compiles quickly
Floating point	No runtime fee

MTBASIC is a true native code compiler. It runs *Byte's* Sept. '81 sieve in 26 seconds; interpreters take over 1400 seconds! Because MTBASIC is multitasking, it can run up to 10 Basic routines at the same time, while displaying ten separate windows. Pop-up/down menus are a snap to implement.

The MTBASIC package includes all the necessary software to run in interpreter or compiler mode, an installation program (so *any* system can use windowing), three demonstration programs and a comprehensive manual.

AVAILABLE for CP/M (Z-80), MS-DOS, and PC-DOS systems.

ORDERING: Specify format when ordering. We accept Visa, MC, checks and COD. Send $49.95 plus $3.50 shipping and handling ($10 overseas) to:

SOFTAID, Inc.

P.O. Box 2412 Columbia, MD 21045-1412
301/792-8096

CIRCLE 78 ON READER SERVICE CARD

C
POWER
••••••
C LIBRARIES
C WINDOWS
••••••
Best You Can Get!

325 Fully Tested Functions

SIX C LIBRARIES

FUNCTIONS YOU DON'T HAVE BUT NEED!
All Source Code. No royalties.

51	screen handling/graphic functions	$49.95
50	cursor/keyboard/data input functions	$39.95
85	superior string functions	$59.95
25	system status & control functions	$29.95
72	utility/DOS/BIOS/time/date functions	$49.95
42	printer control functions	$29.95

••••••
RICHLY COMMENTED
EASY TO LEARN
EASY TO MODIFY
••••••
NO MATTER WHAT ELSE YOU HAVE
GET THESE!!
••••••
ANY 3 LIBRARIES $69.95
ALL 6 LIBRARIES $99.95

50 MOST NEEDED FUNCTIONS
$49.95
••••••••
POWER WINDOWS

PROFESSIONAL WINDOW MANAGEMENT
OVERLAYS, BORDERS,
POPUP MENUS, HELP WINDOWS,
STATUS-LINE, COLOR HIGHLIGHTING,
AND MORE!!!
C WINDOWS: COMPLETE SOURCE CODE $99.95
••••••

ALL LIBRARIES PLUS WINDOWS $149.95

Entelekon
SOFTWARE SYSTEMS

ENTELEKON 12118 KIMBERLEY
HOUSTON, TX. 77024 (713)-468-4412

VISA•MASTERCARD•CHECK

CIRCLE 50 ON READER SERVICE CARD

NGS FORTH

A FAST FORTH, OPTIMIZED FOR THE IBM PERSONAL COMPUTER AND MS-DOS COMPATIBLES.

STANDARD FEATURES INCLUDE:

- 79 STANDARD
- DIRECT I/O ACCESS
- FULL ACCESS TO MS-DOS FILES AND FUNCTIONS
- ENVIRONMENT SAVE & LOAD
- MULTI-SEGMENTED FOR LARGE APPLICATIONS
- EXTENDED ADDRESSING
- MEMORY ALLOCATION CONFIGURABLE ON-LINE
- AUTO LOAD SCREEN BOOT
- LINE & SCREEN EDITORS
- DECOMPILER AND DEBUGGING AIDS
- 8088 ASSEMBLER
- GRAPHICS & SOUND
- NGS ENHANCEMENTS
- DETAILED MANUAL
- INEXPENSIVE UPGRADES
- NGS USER NEWSLETTER

A COMPLETE FORTH DEVELOPMENT SYSTEM.

PRICES START AT $70

NEW◆HP-150 & HP-110 VERSIONS AVAILABLE

NEXT GENERATION SYSTEMS
P.O.BOX 2987
SANTA CLARA, CA. 95055
(408) 241-5909

CIRCLE 61 ON READER SERVICE CARD

manipulation as string search, obtaining substrings, string length, and concatenation. Predicates to convert between strings and other data types are also available, including conversions with terms, atoms, integers, floating point reals, and lists. Each predicate is capable of performing a two-way transformation, depending on which argument is instantiated.

Thirty-two external counters are provided and can be accessed by all parts of a program at any recursive level. Four predicates are provided to initialize, increment, decrement, and recall counters. Other predicates count the number of atoms in a list, convert from lists to atoms, and convert from atoms to lists.

Arity/PROLOG implements versatile I/O facilities. The *keyb(Ascii,Scan)* predicate allows the PROLOG interpreter to scan any codes generated by pressing any of the IBM PC keyboard function keys and cursor control keys on the numeric hexpad. Combinations of these keys with the control key and alternate keys are also scanned.

There are also predicates to check screen cursor position and to move the cursor to a specified location. Screen brightness can be controlled, and the screen will display underlined characters, reverse video and blinking characters, and blinking underlined and blinking reverse video characters. Controlling the scrolling of a specific screen area is also possible.

Both sequential and random access files are supported. Five I/O modes exist: read, write, append, read-or-write, and read-or-append. The latter two modes are involved with random access files. The *SEEK* predicate is used to locate information stored in random access files. Arity/PROLOG also provides predicates to redirect I/O and perform bytewise port I/O.

Accessing the MS-DOS command interpreter also is possible. This temporarily suspends the action of the PROLOG interpreter. Typing *Exit* in MS-DOS takes you back to PROLOG. A wide variety of MS-DOS file manipulation from within Arity/PROLOG can also be performed, including directory listing; making, choosing, and removing directories; deleting and renaming files; accessing and setting file attributes; and accessing the system clock and date.

I was able to run all the benchmark programs with Arity/PROLOG. The interpreter ran a close third to PROLOG-86 and PROLOG-1's interpreters, and its Tower of Hanoi screen output speed was faster than PROLOG-1's.

Automata Design Associates
ADA PROLOG

This product comes from Automata Design Associates. It is available in many versions, ranging from a low-cost introductory model to a large virtual memory system model. The latter incorporates mass storage devices as memory extensions. The language was developed using C.

ADA PROLOG uses a ?- question prompt, making the system more accessible to answering questions. To assert any fact from the keyboard, the *assert()*, *asserta()*, and *assertz()* predicates must be used.

This product supports all PROLOG data types, including floating point numbers. A special, large memory language version supports the 8087 chip. Several mathematical functions are offered, including transcendental, trigonometric, and conversion. *AND, OR, XOR*, and bitwise negation operations are also offered.

ADA PROLOG has no built-in editor, but it offers an *EXEC* function that allows the execution of other programs from within PROLOG, assuming there is enough memory. Thus your favorite text editor can be invoked to change any PROLOG program. The *reconsult* predicate can be called to update the fact data base to reflect changes made during the editing process.

Many new predicates included in ADA PROLOG perform a variety of routine tasks, including versatile I/O operations. ADA PROLOG supports the MS-DOS 2.0+ directory structure, I/O redirection, and the predicates to perform file I/O for atoms, characters, strings, and numbers.

Formatted floating point output is available using the C language format rule. The *UPDATE* function permits updating of the resident memory data base by storing it on the disk. Six optional I/O modes are available when files are opened. Both sequential and random access file I/Os are supported, including append and update options.

ADA PROLOG's modularity is similar to Modula-2's import and export facilities. Thus, PROLOG programmers can divide tasks and create nested libraries using treelike structured domains. Each domain can import and export predicates between multiple-subdomains and a single-parent domain. Like MS-DOS structured file directories, domains also can be created, removed, and selected. Import and export predicates allow the interaction of domains.

All benchmark tests were run on ADA PROLOG. The results show the current implementation to be the slowest of all those reviewed.

ConIX™

NOW ONLY $79.95!

If you think you're missing out on innovative software developments because nobody is writing for CP/M™-80, take a look at us. We've adapted UNIX™ features to CP/M like never before, and with the kind of professional, quality-controlled product that you deserve. That product is none other than the critically acclaimed ConIX Operating System.

ConIX can provide any 48K+ CP/M-80 or compatible system with I/O Redirection and Pipes (uses memory or disk), perfected User Areas, Command and Overlay Path Searching, Auto Screen Paging, 8Mb Print Buffering, 22 new SysCalls, Function Keys, "Virtual" disk system, Archiver (saves over 50% disk), extensive command language, 300+ variables, 100+ commands, pull-down menu, and much more! Uses as little as 1/2K RAM! Runs with CP/M for true data and software compatibility. Installs easily without any system mods!

The ConIX package lists at $165 and has been advertised and sold internationally to many enthusiastic customers since October 1983. As a special limited offer, we've lowered the price of the complete ConIX system by 50% to only $79.95! Don't miss this opportunity to bring your 8-bit micro back into the software revolution. Order your copy of ConIX today!

Price includes manual, 8" disk, and user support. 5¼" conversions available. Contact your local dealer, or buy direct and add shipping: $4.50 UPS, $10 Canada, $25 overseas. NY residents add sales tax.

Computer Helper Industries Inc.
P.O. Box 680 Parkchester Station, NY 10462
Tel. (212) 652-1786 (for information/orders)

"We're helping your computer work better for you!"

UNIX: AT&T Bell Labs, CP/M: Digital Research, ConIX: Computer Helper Ind.

CIRCLE 10 ON READER SERVICE CARD

NEW FOR THE MACINTOSH

PRESENTING THE MEGAMAX C COMPILER

FEATURING:
• IN-LINE ASSEMBLY • ONE PASS COMPILATION • SUPPORT OF DYNAMIC OVERLAYS • FULL ACCESS OF MACINTOSH TOOLBOX ROUTINES • AND MUCH MORE...

DEVELOPMENT SYSTEM PACKAGE INCLUDES:
• FULL-SCALE IMPLEMENTATION [K&R] C COMPILER • THE STANDARD C LIBRARY • ROM ROUTINES LIBRARY • LINKER • LIBRARIAN AND DOCUMENTATION...

DEALER AND USER GROUP INQUIRES INVITED

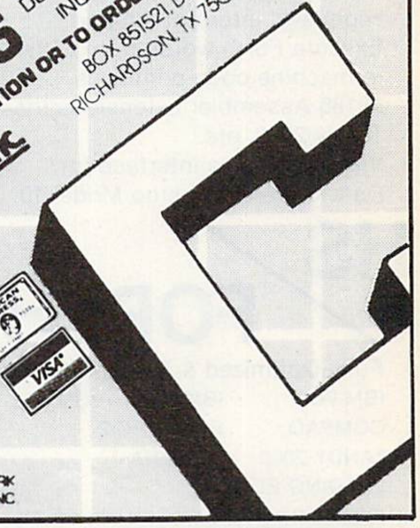
$299.95

FOR MORE INFORMATION OR TO ORDER CALL OR WRITE:

Megamax, Inc.

BOX 851521, DEPT. 1
RICHARDSON, TX 75085-1521

(214) 987-4931



MACINTOSH IS A REGISTERED TRADEMARK OF APPLE COMPUTER INC

CIRCLE 13 ON READER SERVICE CARD

OASYS TOOLKIT SNAP-SHOT

C-68000

CROSS AND NATIVE OPTIMIZING COMPILERS

FOR 68000/10 (and 68020 SOON)

OASYS offers a *"ONE STOP SHOPPING"* service for software developers in need of proven 8-, 16- and 32-bit cross and native tools for Unix and non-Unix 68000, 8086 and 32000 systems. Our critically acclaimed and widely used 68000 tool kit offers high quality, reliable, cost-effective tools.

The OASYS 68000 tool kit consists of Green Hills compilers (C, Pascal and FORTRAN), our own M68000 Macro Assembly Development package, and dozens of other OASYS compatible support tools. Simply stated, we beat the competition on price, speed and tightness of emitted code.

C-68000/10

- Full K & R with Western Electric and Berkeley extensions
- Complete run-time library available as source. No royalty if passed on.
- Supports DEC & IEEE Floating Point
- Integrated optimizer: 30% tighter code than Portable C; 4 times faster
- Generates M.I.T. or EXORmacs assembly source code
- Interfaces to all OASYS tools and Pascal, FORTRAN and PL/M-68K compilers
- Ideal for cross development of boards with no OS, a kernel OS (e.g. VRTX, PSOS, MTOS), or Unix based 68000's

68000/10 Assembly package

- EXORmacs compatibile Macro Assembler, Linker, Librarian, and Cross Reference Utility
- Generates S-records and a.o.
- PIC and reentrant code
- Used 2 years in house
- Over 3,000 sold to date
- Runs on VAX, Prime, PDP-11, 68000's, 8086/88 (PC)
- Written entirely in C

Coming soon

- 68020 C and Cross assembler

Other tools

- Symbolic C Source Debugger
- 68000 Simulator & Disassembler
- C Linecount and Time Profiler Utility (CLUE™)
- LINT for VAX/VMS
- Check Out compiler (SAFE-C™)
- Communications tools

OASYS
60 ABERDEEN AVENUE
CAMBRIDGE, MA 02138
(617) 491-4180

C68K-11/84

CIRCLE 70 ON READER SERVICE CARD

NEW FEATURES

(Free update for our early customers!)

- Edit & Load multiple memory resident files.
- Complete 8087 assembler mnemonics.
- High level 8087 support. Full range transcendentals (tan, sin, cos, arctan, logs and exponentials) Data type conversion and I/O formatting.
- High level interrupt support. Execute Forth words from within machine code primitives.
- 80186 Assembler extensions for Tandy 2000, etc.
- Video/Graphics interface for Data General Desktop Model 10

HS/FORTH

- Fully Optimized & Tested for:

IBM-PC	IBM-XT	IBM-JR
COMPAQ	EAGLE-PC-2	
TANDY 2000	CORONA	
LEADING EDGE		

(Identical version runs on almost all MSDOS compatibles!)

- Graphics & Text (including windowed scrolling)
- Music - foreground and background includes multi-tasking example
- Includes Forth-79 and Forth-83
- File and/or Screen interfaces
- Segment Management Support
- Full megabyte - programs or data
- Complete Assembler (interactive, easy to use & learn)
- Compare

BYTE Sieve Benchmark		jan 83
HS/FORTH 47 sec	BASIC	2000 sec
w/AUTO-OPT 9 sec	Assembler	5 sec
other Forths (mostly 64k)		70-140 sec

FASTEST FORTH SYSTEM AVAILABLE.

TWICE AS FAST AS OTHER FULL MEGABYTE FORTHS!

(TEN TIMES FASTER WHEN USING AUTO-OPT!)

HS/FORTH, complete system only: $250.

Visa Mastercard

Add $10. shipping and handling

HARVARD SOFTWORKS

PO BOX 69
SPRINGBORO, OH 45066
(513) 748-0390

CIRCLE 47 ON READER SERVICE CARD

Chalcedony PROLOG V

Put out by Chalcedony Software, this implementation adheres closely to the Clocksin and Mellish definition. The main benefit of such an implementation is that a novice PROLOG programmer need not worry about language extensions. PROLOG V is a good package to accompany the Clocksin and Mellish text and is certainly affordable.

In PROLOG V, the user uses the > prompt to enter facts or ask questions in a simple and straightforward manner. The [F2] key on the IBM PC keyboard can be used to recall the last typed line.

The PROLOG V implementation does not support floating point numbers. The manual explains that PROLOG is hardly viewed as a number-crunching language and deems the floating point number support omission as minor. PROLOG is more involved in fact manipulation. PROLOG V implements the Clocksin and Mellish data manipulation, I/O, and program tracing and debugging.

Two problems were encountered while running the benchmark tests for this product. The PROLOG V determination allows tracing through decisions made by the compiler, but the manual does not explain how to turn this function off. In addition, stack overflow occurred during many tests.

Expert Systems International PROLOG-1

Expert Systems International's PROLOG-1 has an implementation that adheres closely to the Clocksin and Mellish definition. PROLOG-1 displays the ?- question prompt, making the system more accessible to user questions. To assert any fact from the keyboard, the *assert()*, *asserta()*, and *assertz()* predicates must be used.

PROLOG-1 supports all PROLOG data types, including floating point numbers. Unlike ADA PROLOG, this implementation does not provide transcendental and trigonometric functions. Conversion between integers and floating point numbers is available, as are rounding and truncating reals.

Built-in predicates are provided for altering and inquiring about internal system states, including execution tracing, user polling, I/O redirection, and error handling. The built-in clause editor is able to add, delete, list, and number predicates and allows for multiline definitions by putting a : prompt at the beginning of every continuation line. An I/O clause that saves definitions is also available.

PROLOG-1 implements versatile file I/O, including the I/O predicates defined by Clocksin and Mellish. While sequential and random files are supported, appending data to files is not. Random access is possible using the *seek_read* and *seek_write* predicates to select specific records. The *getbyte* predicate reads binary files, returning the ASCII code number of the byte read.

PROLOG-1 also allows for deleting and renaming files and making inquiries about the existence of files. The interpreter can be suspended, causing a break but preserving the context. A break can be achieved by using the break predicate or interrupts or through the occurrence of certain errors. Program abortion and restart can be achieved in the same ways.

All benchmark tests were run with PROLOG-1. The results show the assembly-language-based implementation to be the second fastest.

Expert Systems International PROLOG-2

The implementation of Expert Systems International's new product was not available for this review, but the manual was.

PROLOG-2 is a superset implementation of PROLOG-1 and uses new nomenclature with substantial enhancements. It is compatible with DEC 10. Two versions of PROLOG-2 are on the distribution disk: one that supports windows and one that does not.

The two primary parts of PROLOG-2 are the interpreter core and the program development satellite facilities. The latter can be customized by the end user. Besides the interpreter, the system core contains a memory allocator, garbage collector, module handler, window handler, system editor, error and help handler, expression evaluator, and string handler. System state predicates are also provided to allow some control of the core system.

Mass storage can act as an extension of the RAM through implementation of a virtual memory. A powerful editor that can alter modules, predicates, and clauses is also supplied. Windows can be used in conjunction with the editor, creating an editing window with predicates for cursor control and an execution window.

A range of values from −8388608 to +8388607 is available through the support of signed 24-bit integers. Floating point reals and strings are also supported. One interesting aspect of PROLOG-2 is that string constants can contain symbols for backspace, form feed, line feed, carriage return, horizontal tab, double quotes, and backslash, all following the convention in C of the backslash being used with another letter. For example, "Hello \ nThere World \ 7" will be displayed with "Hello" on one line and "There World" on another, followed by a beep (ASCII 7 is the code for the bell). Strings also are supported in PROLOG-2, C Sick?

PLZ is the cure!

Introducing a native code PLZ compiler for the 68000, featuring:

- ☐ Complete PLZ language, including structure assignment and comparison
- ☐ Fully compatible with Zilog Z80, Z8000 PLZ
- ☐ Ideal for embedded, ROM based systems
- ☐ Strongly typed
- ☐ Data types include signed and unsigned byte, word and longword
- ☐ All of the protection of Pascal, with the flexibility of C
- ☐ Inherently more portable than either Pascal or C
- ☐ Easy for Pascal or C programmers to learn
- ☐ Fully compatible with the CP/M-68K C library

Requires CP/M-68K. Other systems and CPU's supported soon.

Package includes:
68000 Compiler-Code generator
User Manual
Springer-Verlag "Report on the Programming Language PLZ/SYS"
One Year free updates

All this for the low introductory price of $75

Add $3 S/H. NJ Residents include 6% sales tax.

KC SYSTEMS
KCSystems
20 Lamington Drive, Succasunna, NJ 07876
(201) 927-9104

CIRCLE 59 ON READER SERVICE CARD

SPARRY BASIC-B COMPILER

1. Floating Point Math
2. Use all 640K of Memory
3. Multiple Data Segments
4. Multiple Code Segments
5. Internal ISAM File support
6. 4 Virtual Screens (Big Windows)
7. Easy Assembly Language Interface
8. Direct System Interrupt Calls
9. A Compatible BASIC Compiler

Req. PCDOS 2.00 + with 128K

Sparry Software Labs

P.O. BOX 632
MILFORD, MA 01757
617-473-5435

( ) Compiler $159
( ) Demo Disk $15

PCDOS is a Trademark of International Business Machine Corp.

CIRCLE 66 ON READER SERVICE CARD

OASYS TOOLKIT SNAP-SHOT

WIZARD
C-8086
CROSS AND NATIVE OPTIMIZING COMPILERS
FOR 8086/87/88/186/286

As part of OASYS' *"ONE STOP SHOPPING"* service for software engineering tools, we are proud to announce the addition of WIZARD C to our integrated collection of more than 50 professional programming tools (e.g. compilers, assemblers, linkers, debuggers, simulators & translators) for M68000, Intel 8086/80186 and NS32000 micros.

WIZARD C benchmarks (against Lattice, Microsoft) prove that it is, by far, the most advanced, full featured, fastest, tightest, optimizing C compiler now available for cross and native (PC) development. Here's why . . .

FEATURES

- Complete K&R implementation plus V7, III extensions
- Supports 8087 Floating Point
- Built-in **LINT**
- Long, medium, short models
- 190 + UNIX III functions — complete run-time library
- In-line assembly allowed
- 100 + extensive warnings/diagnostics
- Intel and Microsoft compatibility at source and object levels
- Written in C — easily ported
- Comprehensive bound documentation
- Supports DOS 2.0, 2.1, IBM/BIOS

SUPPORT TOOLS

- Symbolic C Source Level Debugger (CDEBUG™)
- 100% Intel compatible structured Macro Cross Assembler, Linker/Locater and Librarian
- 8086 Simulator
- Floating point math package (40 + functions)
- C Time Profiler (CLUE™)
- Checkout compiler (SAFE-C™)
- Comm. utilities for up/down loading to MDS, TEK, Microtek

AVAILABILITY

NATIVE: PC/XT/AT using MS/DOS, PC/DOS (Xenix soon)

CROSS: VAX/VMS, Bsd 4.1, 4.2, III, V; 8086's, 68000's (All Unisoft III, V; ports); Callan, Masscomp, Sun, Pyramid, dozens more . . .

Call for pricing, OEM, Site, Corporate, Source and Maintenance licensing information.

OASYS
60 ABERDEEN AVENUE
CAMBRIDGE, MA 02138
(617) 491-4180

W86CH-11/84

CIRCLE 71 ON READER SERVICE CARD

RUN/C:™
Finally,
a C Interpreter

Available NOW for only **$149.95!**

Finally, a painless introduction to the C language. With **RUN/C: The C Interpreter** you can create and run C language programs in an environment as easy to use as BASIC.

RUN/C is C for the rest of us. It is a robust implementation of standard K&R. **RUN/C** is for both the beginner and professional.

RUN/C includes full floating point, 8087 support, structures, unions, casts and more than 100 built-in C functions.

With **RUN/C** you get all this with a command structure modeled after BASIC's using familiar terms such as EDIT, RUN, LIST, LOAD, SAVE, TRON, SYSTEM, etc.

Since **RUN/C** is a true interpreter it means that C programs can be written, tested and run within a single protected environment. It is a teaching tool and a source code debugger.

Here's more good news . . .

- Great documentation: a 400-page, easy-to-read manual filled with executable programs
- Array-index and pointer bounds checking
- Variable-trace and dump diagnostics PLUS an integral program profiler
- Full buffered and unbuffered file I/O
- Printer and asynch support
- Forking to your favorite full screen editor with automatic return to **RUN/C** with your edited program
- System Requirements: IBM® PC or compatible with PC-DOS 2.0 or MS™-DOS 2.0 or greater with ANSI.SYS.

Get things right the first time with **RUN/C: The C Interpreter.™**

For immediate delivery or more information, call:

1-800-847-7078
(in N.Y. **1-212-860-0300**)

or write: Lifeboat Associates™
1651 Third Avenue
New York, NY 10128

RUN/C is a trademark of Age of Reason Co.

CIRCLE 85 ON READER SERVICE CARD

as are predicates for concatenating, inserting, deleting, and extracting strings.

Each part of the satellite system is a module. There are three types of modules: data, library, and programs. Each can use virtual memory and, like the USCD p-system, can be swapped in and out of memory as required. Modules can also contain external code written in other languages, assuming the interface conventions are followed. The data modules contain PROLOG data base information (that is, clauses). Program modules contain unalterable and private executable programs. Library modules are similar to program modules except that individual predicates can be loaded into the memory as needed.

PROLOG-2 implements a versatile and powerful low-level I/O. Streams are used to perform all I/O operations and must be created, opened, and then closed when no longer needed. Some streams, including special devices with names such as printer, reader, punch output, screen, keyboard, mouse, and glass-tty, are created automatically but still need to be opened.

There are four stream types: files, devices, user-defined streams, and windows. File streams are employed for I/O with DOS disk files. Sequential and random access I/O are supported. Device streams are related to the special devices previously mentioned. User-defined streams are required to handle I/O from nonstandard devices. Window streams are similar to the screen, but I/O is normally confined to a screen portion (that is, window boundaries). PROLOG-2 has several predicates that set up, manipulate, and manage windows.

Solution Systems PROLOG-86

Two versions of this Solution Systems product (written by Micro-AI) are available. One is designed for the novice PROLOG student. The other is primarily a development system. The implementation is close to the Clocksin and Mellish definition, with a few changes. The distribution disk includes Dean Scholobom's tax program.

PROLOG-86 allows for the user to enter rules, questions, and commands. Rules are asserted by the period. Questions can be asked either by typing ?-, followed by a question that ends with a period, or by typing the question and ending it with a question mark. Commands allow for the execution of a variety of statements, mostly I/O, without having to end with the usual PROLOG "Yea" or "Nea."

PROLOG-86 allows the user to invoke a text editor. This can be specified by a *set* statement (for example, *set EDITOR = <your editor>*) in the CONFIG.SYS file, which sets up the environment for MS-DOS during the boot stage, or by using the PROLOG-86 predicate *set_editor* to make the necessary selection. Three predicates are involved. The *ef* predicate allows for a text file but does not force the PROLOG-86 interpreter to read it. The *ed* predicate allows the editing of a text file containing PROLOG definitions. As the editing ends, the interpreter reads the file to update the information in the data base. Errors are flagged and the user is given an option to reedit or abort and restore the previous definitions. This is a powerful and interactive way of editing. The third predicate, *em*, removes (unloads) the current definitions in the memory before going into the editing process.

PROLOG-86 supports floating point reals and transcendental, trigonometric, absolute, ceiling, floor, and square root functions. Theses functions are implemented as predicates but still act like functions in the sense that each function is not capable of implicitly becoming its reverse function.

The 8087 numeric coprocessor chip is supported by a special PROLOG-86 version that is available on the distribution disk. Basic bitwise operations are also implemented, including bitwise *AND, OR, XOR,* left and right shift, and complements. PROLOG-86 defines *if-then* and *if-then-else* constructs using - >. This allows decision making and the selection of alternate action.

PROLOG-86 also presents a number of additional functions. Among them is *change_case(L,U)*, which allows for a two-way upper-lower case conversion. If the *L* variable is instantiated (initialized in PROLOGese), then the *U* returns the upper case atom. On the other hand, when *U* is instantiated the *L* variable returns the lower-case atom.

Other PROLOG-86 predicates allow the user to perform screen cursor control, including clearing lines and screen and cursor positioning. The *concat(List,Atom)* predicate concatenates the list elements and produces a single atom. Its reverse predicate is *name(Atom,List)*. The *length(List,Num)* predicate returns the number of atoms in a list.

The I/O operations supported by PROLOG-86 have the ability to invoke the MS-DOS command shell. In addition, the interpreter can be temporarily suspended so the user can go back to the operating system. The *exit* command issued in the operating system resumes the interpreter. MS-DOS structured file paths are not supported in this version, but deleting and renaming files is possible. The directory command allows the user to inspect active directories.

All benchmark tests were run with PROLOG-86. The results show it to be the fastest.

Poor Person Software
Introduces

Write-Hand-Man

Desk accessories for CP/M

Write-Hand-Man lets you take notes, check phone numbers, make appointments, and countless other tasks without leaving Wordstar, dBase, Multiplan, or any other application. Enter **Write-Hand-Man** with a single keystroke and choose the program you want. When you leave **Write-Hand-Man,** your application continues normally.

$49.95 plus tax delivers **Write-Hand-Man** and 4 companion programs; **Notepad, Phonebook, Calendar,** and **Termcomm.** User written programs are easily added. All you need is M80 or some other LINK-80 compatible assembler.

Other CP/M products available from **Poor Person Software: Poor Person's Spooler** ($49.95), **Poor Person's Spelling Checker** ($29.95), **Poor Person's Spread Sheet** ($29.95), **Keyed Sequential Files** ($39.95), **Poor Person's Menus** ($29.95), **aMAZEing Game** ($29.95), **Window System** ($29.95), **Crossword Game** ($39.95), **Mailing Label Processor** ($29.95). Shipping included.

All products available on IBM 8 inch and Northstar 5 inch disks. Other 5 inch formats add $5 handling charge. No credit cards.

Poor Person Software
3721 Starr King Circle
Palo Alto, CA 94306
tel 415-493-3735

CP/M is a registered trademark of Digital Research

CIRCLE 67 ON READER SERVICE CARD

WRITE

The Writer's Really Incredible Text Editor lives up to its name! It's designed for creative and report writing and carefully protects your text. Includes many features missing from WordStar, such as sorted directory listings, fast scrolling, and trial printing to the screen. All editing commands are single-letter and easily changed. Detailed manual included. Dealer inquiries invited. WRITE is $239.00.

BDS's C Compiler

This is the compiler you need for learning the C language and for writing utilities and programs of all sizes and complexities. We offer version 1.5a, which comes with a symbolic debugger and example programs. Our price is (postpaid) $130.00.

Tandon Spare Parts Kits

One door latch included, only $32.50.
With two door latches $37.50.
Door latches sold separately for $7.00.

All US orders are postpaid. We ship from stock on many formats, including: 8", Apple, Osborne, KayPro, Otrona, Epson, Morrow, Lobo, Zenith, Xerox. Please request our new catalog. We welcome COD orders.

Workman & Associates
112 Marion Avenue
Pasadena, CA 91106
(818) 796-4401

WORKMAN & ASSOCIATES
USER-FRIENDLY SOFTWARE

CIRCLE 68 ON READER SERVICE CARD

C Programmers: File System Utility Libraries

Source Code Included, No Royalties, Powerful & Portable.

BTree Library **$75.00**

- High speed random and **sequential** access.
- Multiple keys per data file.
- Up to 16 million records per file.
- Full documentation and example programs included.

ISAM Driver **$40.00**

- Works with the BTree Library.
- Greatly speeds application development.
- Combines ease of use of database manager with flexbility of programming language.
- Supports multi key files and dynamic index definition.
- Very easy to use; fully documented; example programs included.

Both products

Are written entirely in K&R C.
Come with complete source code.
Are free of any royalty charges.

+ $3.00 Shipping & Handling Charge.

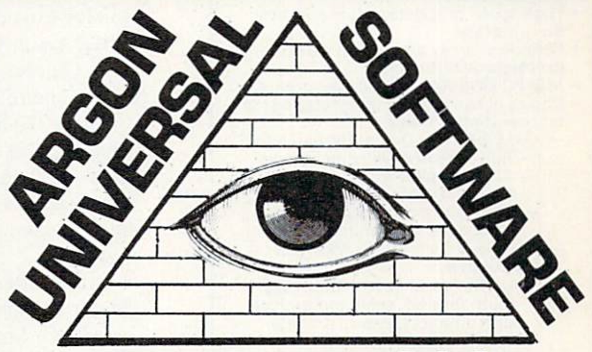
For more information call:

softfocus

1277 Pallatine Drive
Oakville, Ontario, Canada
L6H 1Z1
(416) 844-2610

Credit cards accepted. *Dealer inquiries invited.*

CIRCLE 80 ON READER SERVICE CARD



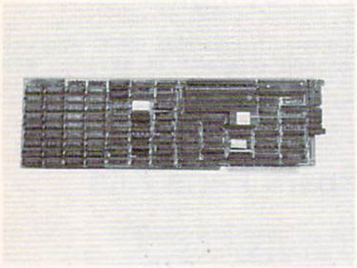
We Beat ALL Prices!

	LIST	OURS
Ascom	195	120
C-Sprite	175	147
Greenleaf communications	185	150
Greenleaf Functions	185	150
Harvard Total Project mgr.	495	320
Instant C	495	400
Lotus	695	375
Lattice	500	275
Lattice Windows	295	217
Pfinish	395	275
Pfix	395	275
Plink	395	275
Pmate	225	150
Pre C	395	275
Run C	150	120
Crosstalk	195	120

253 10th AVE.
NEW YORK, N.Y. 10001 U.P.S. & FedX 212-989-7167

CIRCLE 120 ON READER SERVICE CARD

68000 CO-PROCESSING For IBM PC, PC/XT and COMPATIBLE SYSTEMS



Now you can add the MOTOROLA 68000 16/32 Bit Processor to your PC via use of the Pro 68 Advanced Technology Co-Processor. Enjoy all of the performance benefits of the 68000 processor without sacrificing your current PC system. Consider these impressive standard features of Pro 68:

- High Speed MOTOROLA 68000 micro processor
- 10Mhz no wait state design (3 times faster than the IBM PC/AT)
- True 16/32 bit technology
- For use on IBM PC, PC/XT or compatible systems
- On board 16 bit parity checked memory, 256K to 1024K
- Two serial I/O ports for multi user interface
- Provisions for the high speed NS32081 math processor
- High speed proprietary dual port host bus interface
- Parallel or array processing via multi processor architecture
- MS/PC DOS RAM disk driver program
- Choice of two popular integrated 16/32 bit operating systems:
  - CPM68K from Digital Research Inc.
    - Full suite of development tools
    - "C" compiler with floats and UNIX I/O library
    - Many third party compatible languages and applications
  - OS9/6800 from MICROWARE Corporation
    - UNIX look alike with multi user/multi tasking, shell, hierarchical disk directory, record and file lock, pipes and filters
    - Full suite of development tools
    - UNIX V compatible "C" compiler
    - Optional languages include BASIC, ISO PASCAL, FORTRAN 77.

Pricing from $1195 includes Pro68 with 256K, OS, and MS/PC DOS RAM disk driver. HSC also manufactures and markets a full line of co-processors and RAM disks for use on Z80 based systems.

DISTRIBUTORS:

Australia-Computer Transition Systems	...03-537-2768
Great Britain-System Science	...01-248-;062
West Germany-DSC International	...089-723-1125
Canada Remote Systems	...416-239-2835

Dealer, Distributor and OEM inquiries invited.

HSC

Hallock Systems Co., Inc.
267 North Main Street
Herkimer, NY 13350
(315) 866-7125

CIRCLE 12 ON READER SERVICE CARD

Logic Programming Associates Programming Logic Systems LPA-PROLOG (Micro-PROLOG)

LPA-PROLOG (or Micro-PROLOG), a product of Logic Programming Associates distributed in the U.S. by Programming Logic Systems, employs a somewhat different syntax from that discussed in the Clocksin and Mellish book. It follows the more limited Micro-PROLOG variable name convention. Variable names must start with X, Y, Z, x, y, or z. The LPA Micro-PROLOG package includes a copy of K.L. Clark and F.G. McCabe's book *Micro-PROLOG: Programming Logic*, published by Prentice-Hall.

LPA Micro-PROLOG supports a wide range of integers (−99999999 to +99999999), floating point numbers, and strings in addition to other data structures. LPA Micro-PROLOG has the *SUM()* and *TIMES()* predicates for doing addition, subtraction, multiplication, and division. Both predicates function in one of three ways:

- Result verification. For example, the *SUM(10 15 25)* succeeds since 25 = 10 + 15, while *SUM(12 1 44)* fails because 44 is not equal to (12 + 1).
- Direct operation. *SUM()* and *TIMES()* are used to add and multiply, respectively. For example, *TIMES(10 5 x)* gives *x* = 50 and *SUM(10 5 x)* gives *x* = 15.
- Inverse operation. *SUM()* and *TIMES()* can be used to perform subtraction and division, respectively. Thus, *TIMES(10 x 50)* gives *x* = 5 and *SUM(100 x 144)* gives *x* = 44.

LPA Micro-PROLOG offers predicates to manipulate and compare strings. The *LESS* predicate compares strings. *STRINGOF* packs list members into strings and vice versa. *CHAROF* returns the ASCII code of a character. LPA Micro-PROLOG also has type-checking predicates, which allow checking for numbers, integers, constants, atoms naming a defined program, and variables. LPA Micro-PROLOG also implements the decision-making *if-then* predicate.

Seven console I/O predicates—including those to read terms, display terms, pretty print, poll the keyboard for a pressed key, and reset the keyboard and type-ahead buffers—are also available. File I/O allows the user to read and write characters and text files. Random access and formatted file I/Os are also supported. LPA Micro-PROLOG allows the console, printer, punch output, reader input, and keyboard to be treated as special I/O devices. Other supported file operations include logging to a new disk, erasing and renaming files, and obtaining the disk directory.

Each LPA Micro-PROLOG module has five components: a name, an export list, an import list, a local dictionary, and the module program. Each module can export relations to other modules or programs. The local dictionary contains the list of all local constants. Modules are saved (*SAVE*) and loaded (*LOAD*) and are entered in a hierarchical, treelike structure similar to the ADA PROLOG structure. LPA Micro-PROLOG has four primitives related to modules: *CMOD*, which

Quintus PROLOG

Quintus PROLOG, a product of Quintus Computer System Inc., has an implementation similar to PROLOG-20, DEC10 PROLOG, and C-PROLOG. It runs on the VAX series under VMS, UNIX, and the UNIX 68000-based Sun-2 workstation from Sun Microsystems Inc.

This system provides both an interpreter and a compiler. It can be interfaced with the EMACS editor and uses windows to edit files and run PROLOG. An on-line help system is available. Quintus PROLOG is menu driven and refers to documentation chapters. To interface with C, the user first prepares the required PROLOG facts and then calls the built-in predicate *load_foreign_files* to load C functions into the running PROLOG system.

Quintus PROLOG supports floating point reals, mathematics, and stream-based I/O. Strings are not supported. Many I/O predicates are implemented to handle different data types and redirection, with improved handling of data base references. Other implemented predicates include *keysort* and *sort* for sorting and statistics to output various execution statistics. This system can access UNIX facilities by using the available UNIX predicate.

PROGRAMMER DEVELOPMENT TOOLS

NEW LISTINGS:

	List	Ours
Brief By Solution Systems	195	**Call**
Megamax C compiler for MacIntosh	295	**239**
PC Lint by Gimpel Software	100	**89**
Prolog-86 by Solution Systems	125	**Call**
Scientific Subroutine Lib *for C* by Peerless	175	**159**

C-terp Complete C Interpreter
Full K&R C interpreter/semi-compiler which can access functions and externals compiled on various C compilers. Comes with a powerful, integrated screen editor providing a complete professional C programming environment.
List Price $300 Our Price **$269**

C LANGUAGE:

Computer Innovations C-86 Compiler	395	**299**
DeSmet C Compiler with Debugger	159	**145**
Lattice C Compiler from Lattice	500	**339**
Lattice C from Lifeboat *Ltd Qty Special*	500	**275**
Mark Williams C Compiler *w/Source Debugger*	495	**429**
Run/C Interpreter by Age of Reason *Sale*	150	**99**
Safe C Standalone Interpreter by Catalytix	Call	**Call**
Wizard C Compiler by Wizard Systems	450	**399**
Xenix Development System by SCO	1350	**1099**

Microsoft C Compiler version 3.0
This entirely new version of Microsoft's C compiler features fast execution and compact code generation, small, medium and large memory models, Xenix compatibility, a linker and a librarian.
List Price $395 Our Price **$339**

OTHER LANGUAGES:

8088 Assembler w/Z-80 Translator by 2500 AD	100	**89**
APL*PLUS/PC by STSC	595	**469**
BetterBASIC by Summit Software	200	**169**
Golden Common LISP by Gold Hill	495	**Call**
Janus/ADA by R&R Software	900	**699**
MASM-86 ver 3.0 *w/utilities* by Microsoft	150	**119**
Modula-2/86 by Logitech	495	**439**
Professional BASIC by Morgan Computing	99	**89**
RM/Fortran by Ryan-McFarland	595	**439**

C UTILITIES:

C Power Paks From Software Horizons	Call	**Call**
C-Sprite *Symbolic Debugger for Lattice C*	175	**159**
c-tree by FairCom	395	**359**
C Utility Library by Essential Software	Call	**Call**
dBC *dBase/C Interface* by Lattice	250	**219**
dBC *with source code*	500	**459**
ESP for C by Bellesoft	Call	**Call**
GraphiC by Scientific Endeavors	250	**209**
Greenleaf C Functions Library *ver 3.0*	185	**139**
Greenleaf Comm Library	185	**139**
Multi-Halo Graphics by Media Cybernetics	250	**199**
PANEL *Screen Designer ver. 6.0* by Roundhill	295	**234**
Pasm86 *Macro Assembler* by Phoenix	295	**259**
Pfinish *Performance Analyzer* by Phoenix	395	**339**
Pmaker *Program Development Manager* by Phoenix	195	**179**
Pre-C *Lint Utility* by Phoenix	395	**339**
Safe C Dynamic Profiler by Catalytix	150	**Call**
Safe C Runtime Analyzer by Catalytix	400	**Call**
Windows For C by Creative Solutions	195	**139**

TURBO PASCAL:

Screen Sculptor by Software Bottling	125	**109**
Turbo ASYNCH by Blaise Computing	100	**89**
Turbo GRAPHICS TOOLBOX by Borland Int'l	55	**49**
Turbo PASCAL *ver 3.0* by Borland Int'l *Sale*	70	**49**
Turbo PASCAL *w/8087 or BCD* *Sale*	110	**89**
Turbo PASCAL *w/8087 & BCD* *Sale*	125	**99**
Turbo POWER TOOLS by Blaise Computing *New*	100	**89**
Turbo TOOLBOX by Borland Int'l	55	**49**
Turbo TUTOR by Borland Int'l	35	**29**
XTC Text Editor by Wendin	99	**89**

OTHER PRODUCTS:

Advanced Trace-86 by Morgan Computing	175	**159**
APL2C by Lauer Software *Interfaces APL to C*	150	**139**
Blaise Tools *for C & Pascal*	Call	**Call**
Btrieve by SoftCraft	250	**199**
Codesmith-86 *Debugger* by Visual Age	145	**129**
Epsilon *Emacs-like Text Editor* by Lugaru	195	**179**
FORTRAN Libraries by Alpha Computer Service	Call	**Call**
Pfix-86 Plus by Phoenix	395	**299**
Plink-86 *Overlay Linker* by Phoenix	395	**299**
Pmate *Macro Text Editor* by Phoenix	225	**159**
Polytron Products *We Carry a Full Line*	Call	**Call**
Profiler by DWB Associates	125	**89**
Xtrieve by SoftCraft	195	**169**

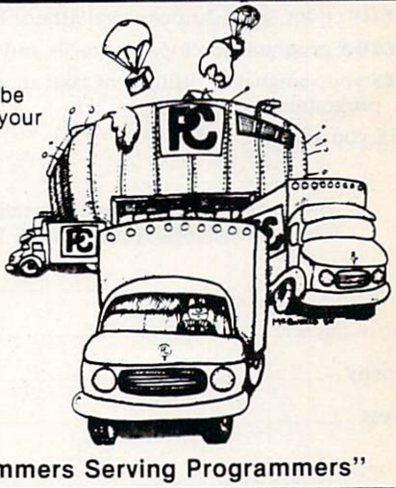
Periscope Symbolic Debugger
by Data Base Decisions
Write-protect memory board and breakout switch allows instant recovery from runaway code. Provides on-line help, windowing, extensive breakpoints, dual monitor support and more.
List Price $295 Our Price **$269**

ADVANTAGE #2

At Programmer's Connection most popular products are always in stock ready to be shipped. Most orders are on their way to you the same day they are placed. We know your time is important, that's why we tell you exactly when you can expect to receive your package. Maintaining an adequate inventory is part of our philosophy of fast, efficient service. Call us — you'll discover that Programmer's Connection delivers products and service without delay.

Discover the advantages of buying from Programmer's Connection:

1. We offer the latest version of a product.
2. Most popular products are in stock ready to be shipped.
3. Receive same manufacturer's support as if buying direct.
4. Experienced professional programmers are on staff.
5. Choose from a large selection of the best software products available.
6. Knowledgeable and courteous sales staff.
7. Significant discounts off of retail prices.
8. No extra charge on prepaid orders, including major credit cards.
9. Reasonable charges for shipping and handling.
10. Toll free services from Canada and the U.S.



"Programmers Serving Programmers"

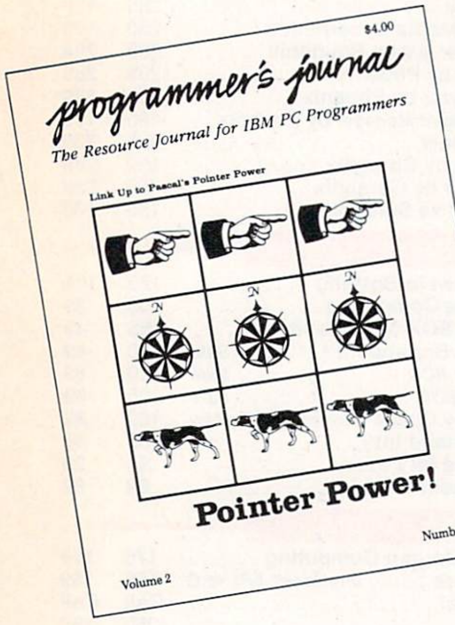
Programmer's Connection
136 Sunnyside Street Hartville, Ohio 44632 (216) 877-3781 (In Ohio)
U.S.; **1-800-336-1166** Canada; **1-800-225-1166**
Call For Our Catalog

Account is charged when order is shipped
Prices are subject to change without notice

CIRCLE 16 ON READER SERVICE CARD

programmer's journal

The Resource Journal for IBM PC Programmers


$4.00
programmer's journal
The Resource Journal for IBM PC Programmers
Link Up to Pascal's Pointer Power
Pointer Power!
Number 5
Volume 2

More than simply a computing magazine, *Programmer's Journal* takes you deep into the art, technology, present and future of programming for the IBM PC. It's a tool, a resource and your voice in the constantly changing environment of computer programming.

PROGRAMMER'S JOURNAL. IT'S YOUR SHOW.

SHOWS you the tricks of the trade, revealing valuable tips and techniques for smoother, faster and more efficient computing . . . with working examples.

PROVIDES you with a source code library, profiles of leading authors and software houses, program listings and unique features on the state of programming art.

SUMS up important articles, books and pertinent research.

BRINGS you timely features, inside information and gives you a forum for sharing *your* ideas.

COVERS codes, applications, legal affairs, new products and more!

REVIEWS program development tools and technological advances.

TALKS your language, letting you take an active part in a programmer-to-programmer dialogue.

SAVES you valuable time!

IBM and Personal Computer are trademarks of International Business Machines, Armonk, New York.

BACK ISSUES

Vol. 1, No. 1: Norton Pascal I/O, Program Style Manual, COBOL, System Analysis, BASIC modules, Rounding error.

Vol. 1, No. 2: Defining character sets, C86 BIOS calls, Copy protection, Pascal code, Input routines, Writing an Assembler.

Vol. 1, No. 3: ASM dumb terminal, C source code, ASM graphics line drawing, Text editors reviewed, Utilities.

Vol. 1, No. 4: Diskette directory via Pascal, MASM WHILE WEND, C/ASM interface, ASM stay resident routine, Structured programming.

Vol. 1, No. 5: Multi-tasking software review, Transporting C Programs, Programmers calculator, dBase bargraphs, Utilities.

Vol. 1, No. 6: ASM printer configuration, DOS patch utility, Hi-speed graphics, Software ownership, dWINDOW review

Vol. 2, No. 1: Freelance consulting, Sorts in C, Pascal text editor source code, Designing national database, Ultra utilities.

Vol. 2, No. 2: Understanding C through Assembly language, Utility routines in C and Assembly, Music in C (using a C music program), Cornucopia of software development tools.

Vol. 2, No. 3: Software development environments, Creating a productive screen handling tool, Generating graphics within windows, Create, edit and play your musical ideas, Linking Assembler to Pascal and Basic.

Vol. 2, No. 4: Writing a custom ROM BIOS for PC-DOS, The future of disk protection, Time-saving MASM Macros, C routines you can purchase.

Vol. 2, No. 5: Scroll control using BASIC and Assembler, RTool, a PC-resident programmer's toolkit, Calling Assembly routines from dBase II, Pascal's pointer power.

Vol. 2, No. 6: Periscope-Building a hardware board for the IBM PC, AT&T surprises all, dBase II to dBase III?, Intel 80286 bugs by FNE, calendar functions for BASIC.

Vol. 3, No. 1: November's Comdex, C - its uses and abuses, Using the C preprocessor, Memory segmentation examples, Index files (part 1), Stuctured programming language, Periscope, a symbolic debugger.

Vol. 3, No. 2: UNIX, Artificial Intelligence, Pointers to functions, Ask/batch file utility, Index files (part 2), Scanning in Turbo Pascal, Calendar date routines in BASIC, Calculate record layouts automatically.

Vol. 3, No. 3: NAPLPS and the professional programmer, Uniforum, Instant C reviewed, Printer control from a batch file, Repositioning the stars in Starfinder, Reducing EXE file size through data positioning, Tools for better programming.

- -

Name ______________________________

Company ______________________________

Address ______________________________

City ______________ State __________ Zip __________

Telephone ______________________________

Send me the following back issues: (Circle the issue(s) desired) $5.50 each U.S., ($6.50 Canada, $8.50 Foreign, U.S. funds) pre-paid only

Vol. 1, No. 1	Vol. 1, No. 5	Vol. 2. No. 3	Vol. 3, No. 1
Vol. 1, No. 2	Vol. 1, No. 6	Vol. 2. No. 4	Vol. 3, No. 2
Vol. 1, No. 3	Vol. 2, No. 1	Vol. 2. No. 5	Vol. 3, No. 3
Vol. 1, No. 4	Vol. 2, No. 2	Vol. 2. No. 6	

☐ 1 year bi-monthly subscription:
$24.00 ($30 Canada, $40 Foreign, U.S. funds only)

☐ Check enclosed ☐ Bill me (subscriptions only)

☐ Charge my Bankcard: ☐ VISA ☐ MC

______________________ Expir. Date ________

Signature ______________________________

Send to:
PROGRAMMER'S JOURNAL
P.O. Box 3000, Dept. EE
Denville, NJ 07834
(503) 345-3043

4520

CIRCLE 75 ON READER SERVICE CARD

returns the name of the current module; *CRMOD*, which creates a new empty module and makes it the current module; *OPMOD*, which enters an existing module and makes it the current module; and *CLMOD*, which closes the current non-root module.

For the benchmark tests, LPA Micro-PROLOG was run using the DEC 10 supervisor. This allowed the programs to run without modifications. The list reversal and Tower of Hanoi tests ran successfully. The Sieve test gave "NO" for an answer instead of the list of primes. The Quicksort test caused stack-overflow errors.

Logicware MPROLOG

This Logicware product is characterized by the numerous predicates it contains and its support of modular programming in a manner very similar to Modula-2. It has a learning tool called LOGIC-LAB and a program development support system (PDSS). MPROLOG has almost enough predicates to warrant a separate review!

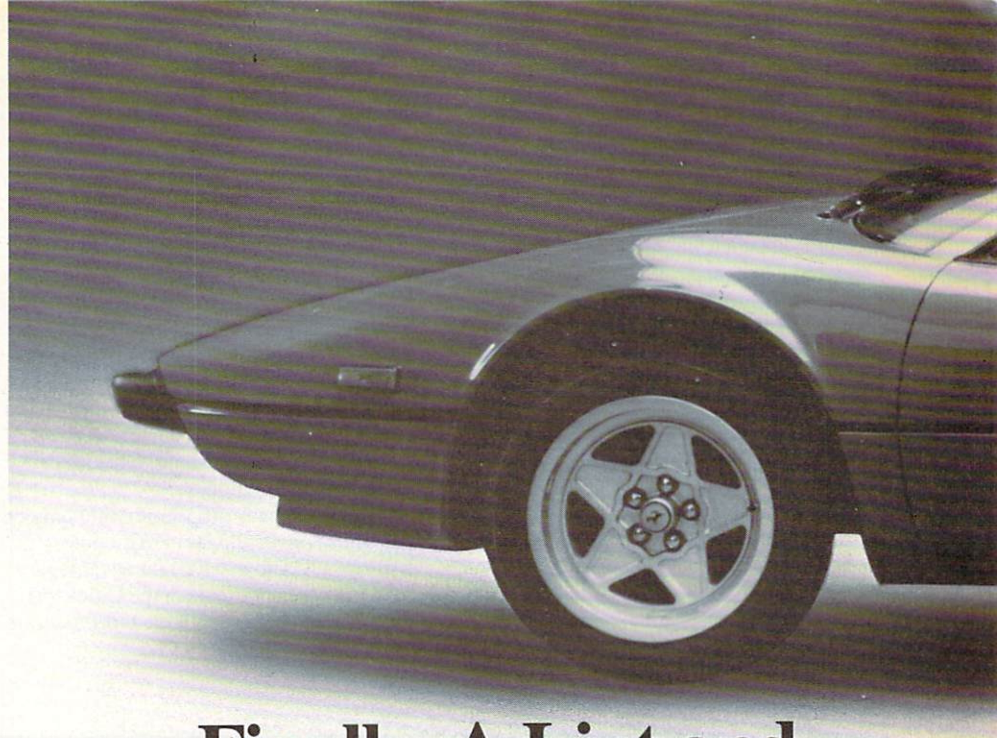
MPROLOG has already been implemented on mainframes and minicomputers, so going to a microcomputer version meant few sacrifices in the PDSS and module implementations. The PDSS allows for the use of *if, and, or*, ":-", ",", ";", or ":", ",", ";" notations in asserting facts. MPROLOG does not support floating point numbers. The conditional *if-then-like->* is supported.

MPROLOG only supports 24-bit integers, with numbers ranging from −8388608 to +8388607.

MPROLOG has many predicates to check the type of expression, including digits, letters, characters, numbers, variables, and nonvariables. Other predicates handle terms and perform tasks such as sorting, keysorting, list counting, and expression comparison. Expression comparison is interesting because it allows the user to inquire about the relation between two expressions. The *compare* predicate is capable of returning relational operators like <=. MPROLOG also has many data base-handling predicates to manage clauses, including adding, renaming, deleting, and inquiring about clauses.

MPROLOG's string manipulation predicates find the string length, extract substrings, concatenate two or more strings, convert between upper and lower cases, convert from ASCII code to characters, and remove blanks from strings.

MPROLOG implements several error-handling and preventing predicates. Preventing predicates can extend the stack, if there is room, to allow more space for recursion. Protection against fatal errors can be turned on or off.



Finally, A Lint and Make for MS™ - DOS

Get the full range of features C programmers working in UNIX™ have come to expect from their Lint and Make utilities. With Pre-C™ you can detect structural errors in C programs five times faster than you can with a debugger. Find usage errors almost impossible to detect with a compiler. Cross-check multiple source files and parameters passed to functions. Uncover interface bugs that are difficult to isolate. All in a single pass. Capabilities no C compiler, with or without program analyzing utilities, can offer. Pre-C outlints Lint, since you can handle analyses incrementally.

Pre-C's flexible library approach lets you maintain continuity across all programs in your shop, whether you use Pre-C's pre-built libraries, functions you already have, or some you might want to buy.

Plus, you're not limited to one particular library. Pre-C keeps track of all the libraries you're using to make sure that code correctly calls them.

With Pmaker™ you can update and track every module in your program. When you make a change in any source or include file, all you do is run Pmaker. It will recompile changed modules and relink your program. With any compiler or linker you choose. Pmaker can update an object module library when one or several of the object modules are changed. You can use Pmaker to handle any task when a change requires several steps.

Pre-C by Phoenix. $395. Pmaker by Phoenix. $195.

Call (1) 800-344-7200.
In Massachusetts (617) 762-5030.

Or, write: Phoenix Computer Products, Corp., 1420 Providence Highway, Suite 115, Norwood, MA 02062.

Phoenix

PROGRAMMERS' PFANTASIES™ BY PHOENIX

Programmers' Pfantasies, Pre-C, and Pmaker are trademarks of Phoenix Computer Products Corporation. MS is a trademark of Microsoft Corporation. UNIX is a trademark of AT&T Bell Laboratories.

CIRCLE 19 ON READER SERVICE CARD

MPROLOG also implements predicates to get the date and time. The predicate *system_state(Indicator, Value)* allows for system control. Indicator specifies what is to be controlled and Value sets its state or value, including turning garbage collection, list notation, mixed and normalized cases, and tail recursion. The indicator also sets the minimum free space that is left in the global, main, and trail stacks that could trigger an error. In addition, the indicator can include CPU time; elapsed processor time for garbage collection; time, date, and storage allocation for the stack; and what tables have been used.

I/O predicates in MPROLOG are extensive and proliferated. I/O redirection is supported. The reading predicates allow a variety of items to be read, such as mathematical and nonmathematical expressions, end of lines, end of files, comments, records, and symbols. Output predicates allow the user to write expressions, print tabs, create new lines, and do spacing. Predicates that inquire about output column location and available space are on the same line.

One of the highlights of MPROLOG is the module implementation. Each module has a name, interface specification, declarations, comments, predicate definitions, and program goals. Modules also can import and export predicates between one another, as in Modula-2. According to the instruction manual, the current version has many features missing in the PC/MS-DOS implementation.

The benchmark programs could only be run after converting them to modules. MPROLOG showed a slight advantage over the other products in the Quicksort test. The other tests reflected its moderate speed. ■

PROLOG products and manufacturers

Arity/PROLOG—$1,950 comp/inter
Arity Corp. $495 interpreter
336 Baker Ave.
Concord, Mass. 07142
(617) 371-1243

ADA PROLOG—$300
Automata Design Associates
1570 Arran Way
Dresher, Pa. 19025
(215) 646-4894

PROLOG V—$69.95
Chalcedony Software
5580 La Jolla Blvd., Ste. 126
La Jolla, Calif. 92037
(619) 483-8513

PROLOG-1—$395
PROLOG-2—$1,895
Expert Systems International
1150 First Ave.
King of Prussia, Pa. 19406
(215) 337-2300

LPA-PROLOG (Micro-PROLOG)—$250
Logic Programming Associates
Programming Logic Systems (distributor)
31 Crescent Dr.
Milford, Conn. 06460
(203) 877-7988

MPROLOG—$725
Logicware
5000 Birch St., West Tower
Ste. 3000
Newport Beach, Calif. 92660
(714) 476-3634

Quintus—Sun: $8,400-$14,280
Vax: $6,600-$21,800
Quintus Computer Systems Inc.
2345 Yale St.
Palo Alto, Calif. 94306
(415) 494-3612

PROLOG-86—$125
Solution Systems
335 Washington St.
Norwell, Mass. 02061
(800) 821-2492

Realia COBOL

What to do while your COBOL programs compile and execute:

1. Wait.
2. Wait some more.
3. Stop waiting. Call Realia.

Patience isn't always a virtue.
Realia COBOL is fast:

Compilation Speed (minutes:seconds)

Lines in Program	Realia COBOL	mbp COBOL	Level II COBOL	R-M COBOL	Microsoft COBOL
1,000	:51	8:33	3:42	5:05	5:11
5,000	3:30	48:07	16:58	*	45:26

*Could not successfully compile the program.

Execution Time Ratio
(Gibson Mix; calculated S-Profile)

Realia COBOL	mbp COBOL	Level II COBOL	R-M COBOL	Microsoft COBOL
1.0	3.6	14.7	21.6	22.3

Sieve of Eratosthenes
0.818 seconds per iteration

All benchmark tests were performed on an IBM PC-XT with 192KB of memory. IBM PC-XT is a registered trademark of International Business Machines Corporation; mbp COBOL, of mbp Software and System Technology; Level II COBOL, of Micro Focus; R-M COBOL, of Ryan-McFarland; and Microsoft COBOL, of Microsoft.

Realia COBOL is written in COBOL. We offer you the tools we use ourselves:

- Our FOLLOW-THE-SOURCE™ interactive symbolic debugger. Works with normal native code.
- A speedy full-screen editor that handles very large files.
- Mainframe IBM VS COBOL compatibility.
- Interfaces to Assembler and C.
- No royalty or run-time fee.
- No limit on program size, up to available memory.
- In our new release, no need to insert the product diskette when you're using a hard disk.

Realia COBOL costs $995. Qualified companies can try it for free. Call us. And ask about our other products, Spacemaker™ and Termulator™.

What are you waiting for?

REALIA inc.
10 South Riverside Plaza
Chicago, Illinois 60606
(312) 346-0642
TELEX: 332979 (REALIA INC)

CIRCLE 76 ON READER SERVICE CARD

STRUCTURE FOR BASIC

With PROFESSIONAL PROGRAMMING ENVIRONMENT

BENDORF ASSOCIATES
6006 S. MAIN
P.O. BOX 5910
ROSWELL, NM
88201
505 347-5701

VISA/MASTERCARD

ELIMINATES LINE NUMBERS

ALLOWS MULTI-LINE CONDITIONALS

PCDOS/MSDOS

WORKS WITH BASICA INTERPRETER COMPILER

FULL ERROR LOGGING

PROGRAM LISTER

LABELED PROCEDURES

MACROS

SUB-ROUTINES

LIBRARIES

$49.95

CIRCLE 6 ON READER SERVICE CARD

MINNESOTA ❄ ❄

SNOBOL4

❄ ONLY $44.95

This exceptional language will amaze you with its versatility. Compatible with main-frame SNOBOL4, use it at work and home. Perform those complicated programming jobs and prototype your exotic ideas quickly. This robust implementation supports large memory model, 8087(if float desired), 32 bit integers, 32K strings. Includes 60 page reference guide and sample programs including ELIZA. You can also get Griswold's et al definitive "green" book or even the source code. Needs IBM PC or 8088/86, ≥128K, IBM or MS DOS.

Guide + 5¼" SSDD diskette	$44.95
Guide + diskette + "green" book	$59.95
"Green" book only	$24.95
Source code and license	$500.00

Prices postpaid. In NY state add sales tax.

☎ (914)271-5855

BERSTIS INTERNATIONAL
P.O. Box 441
MILLWOOD, NY 10546 USA

CIRCLE 88 ON READER SERVICE CARD

Hyperon Software

Specializing in innovative programming tools.

- Complete documentation and C-source provided (presently DOS only).
- Reasonable prices.
- High quality and good performance.

Products currently available:

C Preprocessor. Features include variables and expressions, loops, and full macros. Price — $39.95.

General purpose editor. Line oriented commands with a screen oriented submode. Command window. Price — $29.95

Order from:

HYPERON SOFTWARE
P.O. Box 3349
Costa Mesa, CA 92628

Enclose check or money order. California residents add 6%.
2532 Orange Ave., Costa Mesa, CA

CIRCLE 51 ON READER SERVICE CARD

NEW!

Advanced Trace86™

Symbolic Debugger & Assembler Combo

- Full-screen trace with single stepping; Even backstepping!
- Write & Edit COM & EXE programs
- Conditional breakpoints (programmable)
- Switch between trace and output screen; Or set up two monitors
- 8087, 80186, 80286, 80287 support
- Write labels & comments on code
- Polish hex/decimal calculator
- and more . . . Priced at $175.00

To order or request more information contact:

Morgan Computing Co., Inc.
P.O. Box 112730, Dallas, TX 75011
(214) 245-4763

CIRCLE 22 ON READER SERVICE CARD

FLOWCHARTER

Interactive EasyFlow is a full-blown, on-screen flowchart processor with advanced features. This is a program dedicated to flowcharts & organization charts, not a general-purpose graphics program that makes you do most of the work.

• automatic line routing • automatic text centering within shapes • charts up to 16 shapes wide by 16 shapes high • three levels of zoom (close-up, normal, wide-angle) • horizontal & vertical scrolling within the chart • 18 standard flowcharting shapes included • user defined shapes easily added • wide charts can be printed in strips • complete, well written manual • works with all printers • requires a PC-DOS or MS-DOS machine with IBM compatible graphics adapter & 256K.

Only **$149.95** + $2.00 s & h.

HavenTree Software Limited
P.O. Box 1093-D
Thousand Island Park, NY 13692
(613) 542-7270 Extension 82

CIRCLE 58 ON READER SERVICE CARD

OPT-TECH SORT™

SORT/MERGE Program for the IBM-PC, XT, AT & Compatibles

Btrieve Files Now Supported

- Written in assembly language for **high performance**
  Example: 4,000 records of 128 bytes sorted to give key & pointer file in 30 seconds. **COMPARE!**
- Sort ascending or descending on up to nine fields
- Ten input files may be sorted or merged at one time
- Supports many file structures & data types
- Filesize limited only by your disk space
- Output file can be full records, keys or pointers
- Can be run from keyboard or as a batch command
- Can be called as a subroutine to many languages
- Easy to use — Fully documented
- **$99** — VISA, M/C, Check, Money Order, COD, or PO
  Quantity discounts and OEM licensing available

To order or to receive additional information write or call

OPT-TECH DATA PROCESSING
P.O. Box 2167 Humble, Texas 77347
(713) 454-7428

CIRCLE 40 ON READER SERVICE CARD

Pascal-to-C Translator: Convert UCSD Pascal, MT+ Pascal, and others to K&R C. Handles nested procedures, intrinsic functions, separately compiled units and modules, all data types including long integers. Requires 256K IBM PC/XT/AT. Send for free samples or send us up to 500 lines of Pascal and we will convert it for you for FREE. Licensed source and executable code $5,000, conversions 25 cents/line.

TGL Inc.
4400 Sulphur Springs Rd.
Corvallis, OR. 97330
(503)-745-7476

CIRCLE 95 ON READER SERVICE CARD

CONTROL SYSTEMS SOFTWARE ENGINEER

A Challenging Opportunity in our Florida Corporate Offices

RS&H offers a challenging position with responsibility for development of systems level and applications level software for various Industrial Process Control applications.

Responsibilities will include involvement in associated hardware, and control strategy design, but a strong systems software background is the primary requirement.

Candidates should have at least three years' experience and a strong working knowledge of at least one computer operating system, assembly language, and "C". Familiarity with various DEC equipment operating systems, "C" and MACRO is particularly desirable. Experience with development of real-time software with some Process Control/Instrumentation background is desired. A BS degree in Computer Science or Engineering is a minimum requirement.

RS&H is a multi-disciplined A/E/P firm with a 43 year reputation for innovative technology applications and professional development. **RS&H** offers an attractive salary and benefits package including a prime Florida location providing a quality of life second to none. Please send your resume in complete confidence, including salary history to **Thomas M. Hills (904) 739-2000.**

REYNOLDS, SMITH AND HILLS
Architects-Engineers-Planners, Inc.
P.O. Box 4850
Jacksonville, FL 32201

RS&H encourages qualified minorities, women, veterans and handicapped individuals to apply.

CIRCLE 79 ON READER SERVICE CARD

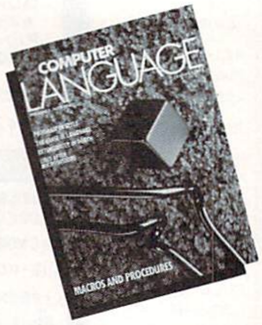
FORTRAN PROGRAMMERS

Discover why
you should be using
F77L
the complete implementation
of the ANSI FORTRAN 77
Standard for the IBM PC and
compatibles.
If you are serious about your
FORTRAN programming, you
should be using F77L.
$477

Lahey Computer Systems, Inc.
31244 Palos Verdes Drive West, Suite 243
Rancho Palos Verdes, California 90274
(213) 541-1200
Serving the FORTRAN community since 1969

CIRCLE 15 ON READER SERVICE CARD

ORDER COMPUTER LANGUAGE BACK ISSUES — WHILE THEY LAST!



PREMIER

The biggest collector's issue
- Basic Becomes a Structured Language—by Kemeny & Kurtz
- Programming in the UNIX Environment
- COBOL: Pride and Prejudice
- Exploring Ada and Modula-2

Exotic Language: SNOBOL
Interview: Charles Moore

OCTOBER '84
- An Implementation Demonstrating C Portability
- The Evolution of ZCPR-Part I
- BATCH—A Powerful IBM "Language"

Exotic Language: PILOT
Interview: Donald Knuth

NOVEMBER '84
- Enhancing Source Code Control under UNIX, Part I
- Natural Language Processing and LISP
- Building Portable Programs

Exotic Language: OCCAM
Interview: Gary Kildall

DECEMBER '84
- Exploratory Programming
- Fred: A Language within Framework
- Six Pascal Compilers Compared

Exotic Language: OMNI
Interview: Bill Godbout

JANUARY '85
- Macros and Procedures
- Extensibility in Forth
- The IIIrd Dimension-Programming in dBASE III

Exotic Language: Transaction Application Language
Interview: Sol Libes

FEBRUARY '85

C Language Special Issue
- Twenty-One C Compilers Compared
- The Standardization of C
- C to Assembly Interface

Exotic Language: MUMPS
Interview: P.J. Plauger

MARCH '85
- Modifying MS-DOS Device Drivers
- Hashing out FORTH with Charles Moore
- Programming Macros in C (Debugging)
- Sol Libes Covers Multiprocessing Systems

APRIL '85
- Recursive Procedures
- Sorting by Dispersion
- COBOL Compiler Analysis
- C on the Macintosh

Exotic Language: APT
Interview: Gordon French

MAY '85

Exclusive Interview: Niklaus Wirth and Donald Knuth
- Twenty-nine BASIC Interpreters and Compilers Compared
- Macros In BASIC

Exotic Language: Clascal

JUNE '85
- Forth: Twithe Curthed, too!
- Porting the UNIX Utilities
- Symphony Command Language

Exotic Language: COMAL
Interview: Ward Christensen

Only a limited quantity of magazines is available, so order today. To receive your back issues, just fill out this coupon and mail it back with a check for $5.00 per issue.

Premier	______	copies X $5.00 = $	______
Oct. '84	______	copies X $5.00 = $	______
Nov. '84	______	copies X $5.00 = $	______
Dec. '84	______	copies X $5.00 = $	______
Jan. '85	______	copies X $5.00 = $	______
Feb. '85	______	copies X $5.00 = $	______
Mar. '85	______	copies X $5.00 = $	______
Apr. '85	______	copies X $5.00 = $	______
May '85	______	copies X $5.00 = $	______
Jun. '85	______	copies X $5.00 = $	______
		TOTAL $	______

Foreign orders: Add $3.00 for airmail.

NAME ______

COMPANY ______

ADDRESS ______

CITY, STATE, ZIP ______

Send payment and coupon to:

COMPUTER LANGUAGE
Back Issues
131 Townsend St.
San Francisco, CA 94107

Creators of COMPUTER LANGUAGE Sponsor the C Expert Forum

C Seminar/ Workshop

September 16-18 Cambridge, Mass

Never before have so many leaders in the C programming field gathered for one event. The C Seminar/ Workshop will be an exciting forum on the latest technical innovations and C language developments. Best of all, you'll experience a practical, hands-on approach in small workshop sessions. All this in the beautiful autumn foliage of New England, just four blocks from Harvard Yard. The C Seminar/Workshop is brought to you by the publishers of COMPUTER LANGUAGE.

The cost for this comprehensive 2½ day event is only $695. Sign up by June 30th and receive a $100 early bird discount.

CURRICULUM

Speakers

Jim Brodie, *ANSI C committee chairman:* Overview of the ANSI Standardization Effort
P.J. Plauger, *author, ANSI C committee secretary:* Programming Style and C
Larry Rosler, *ANSI C language chairman:* Language Standardization Issues
Tom Plum, *author:* Efficiency of C Programs
Heinz Lycklama, */usr/group UNIX chairman:* UNIX Perspective on C
Leor Zolman, *compiler writer:* Porting C Programs between Operating Systems
Robert Ward, *C User's Group coordinator:* Structured Methods of Debugging C

Workshops

(Subject to change based on availability)

Seminar participants will be able to choose four from this list:

Debugging Techniques
Interpreters in a Development Environment
Programming for Portability
Efficient Code Generation
Cross Compilers
Network Data Base Theory and C
Object-File Formats for UNIX Systems
Philosophy and Methodology of Benchmarks
ANSI Standards: Questions & Answers
Code Readability and Organization
Asynchronous Communications
Writing Extensions to C
C / UNIX System Subroutine Interfaces
Porting C between CP/M, MS-DOS, and UNIX

C Seminar/Workshop Registration Form

Please enroll me in the C Seminar:

☐ Early Bird $595 (pay by 6/30/85)
☐ Single $695
☐ Multiple
(3 or more enrollments get $100 discount)
☐ I do not wish to enroll at this time but please send me more information.

Method of Payment:

☐ Check Enclosed
☐ Bill My Company

Make check payable to:
C.L. Publications Inc.

Name & title ______________________
Name & title ______________________
Name & title ______________________
Company ______________________
Address ______________________
City, State, Zip ______________________
Phone ______________________

EA85

COMPUTER LANGUAGE Seminar
131 Townsend St.
San Francisco, Calif. 94107
(415) 957-9353

CIRCLE 116 ON READER SERVICE CARD

ADVERTISER INDEX

	PAGE NO.	CIRCLE NO.
AGS Computers Inc.	1	7
Amber Systems	81	2
Arity Corp.	18	48
Automata Design Assoc.	55	1
Awareco	83	5
BD Software	22	4
Bendorf & Associates	109	6
Berstis International	109	88
Blaise Computing Inc.	6	3
Borland International	cover IV	9
C Ware/DeSmet	78	11
CCA Uniworks Inc.	87	60
Chalcedony Software	8	49
Computer Helper Industries	99	10
COMPUTER LANGUAGE C Seminar	111	116
Creative Programming	21	52
Creative Solutions	76	8
Ecosoft	73	17
Entelekon	97	50
Essential Software Inc.	89	33
ExperTelligence	67	30
Exsys Inc.	39	54
Franz Inc.	63	55
Gimpel Software	59	—
Gimpel Software	73	—
Gracon Services Inc.	40	56
Greenleaf Software Inc.	54	44
HSC Inc.	104	12
Harvard Softworks	100	47
Haventree Software	109	58
Human Computing Resources Corp.	70	93
Hyperon Software	109	51
Inmos Corp.	84	32
KC Systems	101	59
KDS Corp.	33	62
Lahey Computer Systems Inc.	109	15
Lattice Inc.	82	18
Level Five Research Inc.	21	64
Lifeboat Associates	77	87
Lifeboat Associates	102	85
LISP Company (The)	33	65
mbp Software & Systems Technology	93	53
MIX Software	41	24
Major Software	73	14
Manx Software Systems	cover III	69
Megamax Inc.	99	13
Microsoft Corp.	3	—
Microsoft Corp.	5	—
Mindbank Inc.	90	63
Morgan Computing Co.	109	22
Mystic Canyon Software	74	57
Next Generation Systems	98	61
Northwest Computer Algorithms	67	39
Oasys	99	70
Oasys	101	71
Op Tech Data Processing	109	40
Phoenix Computer Products Corp.	107	19
Plu Perfect Systems	56	35
Polytron Corp.	34	34
Poor Person Software	103	67
ProCode	10	73
Programmer's Connection	105	16
Programmer's Journal	106	75
Programmer's Shop (The)	68	84
Programmer's Shop (The)	94	117
Programming Logic Systems	55	21
Quintus Computer Systems Inc.	48	20
Rational Systems Inc.	58	72
Realia Inc.	108	76
Relational Database Systems Inc.	9	77
Reynolds, Smith, & Hills	109	79
SLR Systems	91	28
STSC	cover II	121
SoftCraft Inc.	79	31
Softaid Inc.	97	78
Softfocus	103	80
Software Horizons	91	25
Softway Inc.	42	29
Solution Systems	75	42
Solution Systems	75	122
Sparry Software Labs	101	66
Speedware	55	41
Spruce Technology Corp.	17	26
SuperSoft	60	74
Systems Guild	10	27
Systems Peripheral Consultants	91	38
TGL Inc.	109	95
Tardis Software	21	82
Technisoft	72	83
Turbo Power Software	53	45
UniPress	63	81
Unitek Technologies Corp.	56	23
Watcom Products Inc.	25	36
Wendin	23	43
Wizard Systems	97	86
Workman & Associates	103	68

The index on this page is provided as a service to our readers. The publisher does not assume any liability for errors or omissions.

BUY AN AD GET A CAT!

(Details to come next month.)

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY CARD

FIRST CLASS PERMIT NO. 27346 PHILADELPHIA, PA USA

POSTAGE WILL BE PAID BY

COMPUTER
LANGUAGE

P.O. BOX 11747
PHILADELPHIA, PA 19101

SUBSCRIBE

COMPUTER
LANGUAGE

Subscribe to **COMPUTER LANGUAGE** today for only $24.95—over 30% savings off the single copy price.

☐ Yes, start my Subscription to COMPUTER LANGUAGE today. The cost is only $24.95 for 1 year (12 issues).

☐ I want to increase my savings even more—send me 2 years (24 issues) of COMPUTER LANGUAGE for only $39.95.

☐ Payment enclosed ☐ Bill me

Name ______________________________

Company ______________________________

Address ______________________________

City, State, Zip ______________________________

Please allow 6–8 weeks for delivery of first issue. Foreign orders must be prepaid in U.S. funds. Canada orders $30.95 per year. Outside the U.S., $36.95/year for surface mail or $54.95/year for airmail.

CI75

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY CARD

FIRST CLASS PERMIT NO. 27346 PHILADELPHIA, PA USA

POSTAGE WILL BE PAID BY

COMPUTER
LANGUAGE

P.O. BOX 11747
PHILADELPHIA, PA 19101

SUBSCRIBE

COMPUTER
LANGUAGE

Subscribe to **COMPUTER LANGUAGE** today for only $24.95—over 30% savings off the single copy price.

☐ Yes, start my Subscription to COMPUTER LANGUAGE today. The cost is only $24.95 for 1 year (12 issues).

☐ I want to increase my savings even more—send me 2 years (24 issues) of COMPUTER LANGUAGE for only $39.95.

☐ Payment enclosed ☐ Bill me

Name ______________________________

Company ______________________________

Address ______________________________

City, State, Zip ______________________________

Please allow 6–8 weeks for delivery of first issue. Foreign orders must be prepaid in U.S. funds. Canada orders $30.95 per year. Outside the U.S., $36.95/year for surface mail or $54.95/year for airmail.

CI75

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY CARD

FIRST CLASS PERMIT NO. 22481 SAN FRANCISCO, CA USA

POSTAGE WILL BE PAID BY

COMPUTER
LANGUAGE

2443 FILLMORE STREET • SUITE 346
SAN FRANCISCO, CA 94115

FREE INFORMATION

Name ____________________

Company ____________________

Address ____________________

City, State, Zip ____________________

Country__________ Telephone number__________

July issue. Not good if mailed after November 30, 1985.

Circle numbers for which you desire information.

1	11	21	31	41	51	61	71	81	91	101	111	121	131	141
2	12	22	32	42	52	62	72	82	92	102	112	122	132	142
3	13	23	33	43	53	63	73	83	93	103	113	123	133	143
4	14	24	34	44	54	64	74	84	94	104	114	124	134	144
5	15	25	35	45	55	65	75	85	95	105	115	125	135	145
6	16	26	36	46	56	66	76	86	96	106	116	126	136	146
7	17	27	37	47	57	67	77	87	97	107	117	127	137	147
8	18	28	38	48	58	68	78	88	98	108	118	128	138	148
9	19	29	39	49	59	69	79	89	99	109	119	129	139	149
10	20	30	40	50	60	70	80	90	100	110	120	130	140	150

I obtained this issue through:
- ☐ Subscription
- ☐ Passed on by associate
- ☐ Computer Store
- ☐ Other __________
- ☐ Retail outlet

Comments ____________________

Attn: Reader Service Dept.

2/7

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY CARD

FIRST CLASS PERMIT NO. 22481 SAN FRANCISCO, CA USA

POSTAGE WILL BE PAID BY

COMPUTER
LANGUAGE

2443 FILLMORE STREET • SUITE 346
SAN FRANCISCO, CA 94115

FREE INFORMATION

Name ____________________

Company ____________________

Address ____________________

City, State, Zip ____________________

Country__________ Telephone number__________

July issue. Not good if mailed after November 30, 1985.

Circle numbers for which you desire information.

1 11 21 31 41 51 61 71 81 91 101 111 121 131 141
2 12 22 32 42 52 62 72 82 92 102 112 122 132 142
3 13 23 33 43 53 63 73 83 93 103 113 123 133 143
4 14 24 34 44 54 64 74 84 94 104 114 124 134 144
5 15 25 35 45 55 65 75 85 95 105 115 125 135 145
6 16 26 36 46 56 66 76 86 96 106 116 126 136 146
7 17 27 37 47 57 67 77 87 97 107 117 127 137 147
8 18 28 38 48 58 68 78 88 98 108 118 128 138 148
9 19 29 39 49 59 69 79 89 99 109 119 129 139 149
10 20 30 40 50 60 70 80 90 100 110 120 130 140 150

I obtained this issue through:
☐ Subscription ☐ Passed on by associate
☐ Computer Store ☐ Other __________
☐ Retail outlet

Comments ____________________

Attn: Reader Service Dept.

2/7

AZTEC

The Most Powerful C

for the IBM AT • MACINTOSH • MS DOS • CP/M-80 • ROM APPLICATIONS IBM PC/XT • APPLE // • CP/M-86 • TRSDOS • CROSS DEVELOPMENT

Why Professionals Choose Aztec C

AZTEC C compilers generate fast, compact code. AZTEC C is a sophisticated development system with assemblers, debuggers, linkers, editors, utilities and extensive run time libraries. AZTEC C is documented in detail. AZTEC C is the most accurate and portable implementation of C for microcomputers. AZTEC C supports specialized professional needs such as cross development and ROM code development. MANX provides qualified technical support.

AZTEC C86/PRO

— for the IBM AT and PC/XT

AZTEC C86/PRO provides the power, portability, and professional features you need to develop sophisticated software for PC DOS, MS DOS AND CP/M-86 based microsystems. The system also supports the generation of ROM based software for 8088/8086, 80186, and 80286 processors. Options exist to cross develop ROM code for 65xx, 8080, 8085, and Z80 processors. Cross development systems are also available that target most micro computers. Call for information on AZTEC C86/PRO support for XENIX and TOPVIEW.

POWERFUL — AZTEC C86/PRO 3.2 outperforms Lattice 2.1 on the DHRYSTONE benchmark 2 to 1 for speed (17.8 secs vs 37.1) while using 65% less memory (5.8k vs 14k). The AZTEC C86/PRO system also compiles in 10% to 60% less time and supports fast, high volume I/O.

PORTABLE — MANX Software Systems provides **real** portability with a family of compatible AZTEC C software development systems for PC DOS, MS DOS, CP/M-86, Macintosh, CP/M-80, APPLE //+, //e, and //c (NIBBLE - 4 apple rating), TRSDOS (80-MICRO - 5 star rating), and Commodore C64 (the C64 system is only available as a cross compiler - call for details). AZTEC C86/PRO is compatible with UNIX and XENIX.

PROFESSIONAL — For professional features AZTEC C86/PRO is unparalleled.

- Full C Compiler (8088/8086 - 80186 - 80286)
- Macro Assembler for 8088/8086/80186/80206
- Linkage Editor with ROM support and overlays
- Run Time Libraries - object libraries + source DOS 1.x; DOS 2.x; DOS 3.x; screen I/O; Graphics; UNIX I/O; STRING; simulated float; 8087 support; MATH; ROM; CP/M-86
- Selection of 8088/8086, 80186, or 80286 code generation to guarantee best choice for performance and compatibility
- Utility to convert AZTEC object code or libraries to Microsoft format. (Assembly + conversion takes less than half the time as Microsoft's MASM to produce MS object)
- Large memory models and sophisticated memory management
- Support products for graphics, DB, Screen, & ...
- ROMable code + ROM support + separate code and data + INTEL Hex Converter
- Symbolic Debugger & Other Utilities
- Full Screen Editor (like Vi)
- CROSS Compilers are available to APPLE //, Macintosh, CP/M-80, TRSDOS, COMMODORE C64, and ROM based 65xx, and 8080/8085/Z80
- Detailed Documentation

AZTEC C86/PRO-AT (configured for IBM AT - options for 8088/8086)	$500
AZTEC C86/PRO-PC/XT (configured for IBM PC/XT - options for 80186/80286)	$500

AZTEC C86/BAS includes C compiler (small model only), 8086 MACRO assembler, overlay linker, UNIX, MATH, SCREEN, and GRAPHICS libraries, debugger, and editor.

AZTEC C86/BAS	$199
AZTEC C86/BAS (CP/M-86)	$199
AZTEC C86/BAS (DOS + CP/M-86)	$299
UPGRADE to AZTEC C86/PRO	$310
C-TREE Database with source	$399
C-TREE Database (object)	$149

CROSS COMPILERS

Cross Compilers for ROM, MS DOS, PC DOS, or CP/M-86 applications.

VAX - > 8086/80xxx cross	$5000
PDP-11 - > 8086/80xxx cross	$2000

Cross Compilers with PC DOS or CP/M-86 hosts are $750 for the first target and $500 for each additional target. Targets: 65xx; CP/M-80; C64; 8080/8085/Z80; Macintosh; TRSDOS; 8086/8088/80186/80286; APPLE //.

AZTEC C68K

— for the Macintosh

For power, portability, and professional features AZTEC C68K-c is the finest C software development system available for the Macintosh.

The AZTEC C68K-c system includes a 68000 macro assembler, a linkage editor, a source editor, a mouse based editor, a SHELL development environment, a library of UNIX I/O and utility routines, full access and support of the Macintosh TOOLBOX routines, debugging aides, utilities, make, diff, grep, TTY simulator with upload & download (source supplied), a RAM disk (for 512K Mac), a resource maker, and a **no royalty** license agreement. Programming examples ar included. (Over 600 pages of documentation).

AZTEC C68K-c requires a 128K Macintosh, and two disk drives (frugal developers can make do with one drive). AZTEC C68K supports the 512K Macintosh and hard disks.

AZTEC C68K-c (commercial system)	$500
AZTEC C68K-p (personal system)	$199
AZTEC C68K-p to AZTEC C68K-c upgrade	$310
Mac C-tree database	$149
Mac C-tree database with source	$399
Lisa Kit (Pascal to AZTEC C68k object converter)	$ 99

AZTEC C65

— for the APPLE //

"...The AZTEC C-system is one of the finest software packages I have seen..." NIBBLE review, July 1984.

The only commercial C development system available that runs native on the APPLE II+, IIc, and IIe, the AZTEC C65 development system includes a full floating point C compiler compatible with UNIX C and other MANX AZTEC C compilers, a 6502 relocating assembler, a linkage editor, a library utility, a SHELL development environment, a full screen editor, UNIX I/O and utility subroutines, simple graphics, and screen functions.

AZTEC C65 (Apple DOS 3.3)	$199
AZTEC C65/PRO (Apple DOS + ProDos) (call for availability)	$350

AZTEC C II/PRO

— for CP/M-80

The first member of the AZTEC C family was the CP/M-80 AZTEC C compiler. It is "the standard" compiler for development on CP/M-80. The system includes the AZTEC C II C compiler, an 8080 assembler, a linkage editor, an object librarian, a full library of UNIX I/O and utility routines, CP/M-80 run time routines, the SMALL library (creates modules less than 3K in size), the fast linker for reduced development times, the ROM library, RMAC and M80 support, library source, support for DRI's SID/ZSID symbolic debugger, and more.

AZTEC C II/PRO	$349
AZTEC CII/BAS	$199
C-TREE Database with source	$399
C-TREE Database in AZTEC object form	$149

AZTEC C80

— for TRSDOS (Radio Shack Model III & 4)

"I've had a lot of experience with different C compilers, but the Aztec C80 Compiler and Professional Development System is the best I've seen." 80-Micro, December, 1984, John B. Harrell III

This sytem has most of the features of AZTEC C II for CP/M. It is perhaps the best software development system for the Radio Shack Model III and IV.

AZTEC C80 model 3 (no floating point)	$149
AZTEC C80 model 4 (full)	$199
AZTEC C80/PRO (full for model 3 and 4)	$299

To order or for information call:

800-221-0440

(201) 530-7997 (NJ and outside U.S.A.). Or write: MANX SOFTWARE SYSTEMS, P.O. Box 55, Shrewsbury, N.J. 07701.

MANX

SHIPPING INFORMATION - Standard U.S. shipment is UPS ground (no fee). In the U.S. one day shipment is $20, two days is $10. Canadian shipment is $10. Two days shipment outside the U.S. is by courier and is freight collect.

TRS 80 RADIO SHACK TRS DOS is a trademark of TANDY.
APPLE DOS MACINTOSH is a trademark of APPLE.

For Technical Support (Bug Busters) call: 201-530-6557

CIRCLE 69 ON READER SERVICE CARD

Speed, Power, Price.
Borland's Turbo Pascal Family.

$69.95

The industry standard. With more than 250,000 users worldwide Turbo Pascal is the industry's de facto standard. Turbo Pascal is praised by more engineers, hobbyists, students and professional programmers than any other development environment in the history of microcomputing. And yet, Turbo Pascal is simple and fun to use!

Jeff Duntemann, PC Magazine: *"Language deal of the century . . . Turbo Pascal: It introduces a new programming environment and runs like magic."*

Dave Garland, Popular Computing: *"Most Pascal compilers barely fit on a disk, but Turbo Pascal packs an editor, compiler, linker, and run-time library into just 29K bytes of random-access memory."*

Jerry Pournelle, BYTE: *"What I think the computer industry is headed for: well documented, standard, plenty of good features, and a reasonable price."*

Portability. Turbo Pascal is available today for most computers running PC DOS, MS DOS, CP/M 80 or CP/M 86. A XENIX verison of Turbo Pascal will soon be announced, and before the end of the year, Turbo Pascal will be running on most 68000 based microcomputers.

High resolution monochrome graphics for the IBM PC and the Zenith 100 computers

$54.95
NEW

Dazzling graphics and painless windows. The Turbo Graphix Toolbox will give even a beginning programmer the expert's edge. It's a complete library of Pascal procedures that include:

—Full graphics window management.
—Tools that will allow you to draw and hatch pie charts, bar charts, circles, rectangles and a full range of geometric shapes.
—Procedures that will save and restore graphic images to and from disk.
—Functions that will allow you to precisely plot curves.
—Tools that will allow you to create animation or solve those difficult curve fitting problems.
and much, much more . . . . .

No sweat and no royalties. You may incorporate part, or all of these tools in your programs, and yet, we won't charge you any royalties. Best of all, these functions and procedures come complete with commented source code on disk ready to compile!

Searching and sorting made simple

$54.95

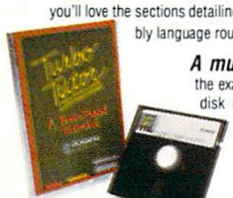
The perfect complement to Turbo Pascal. It contains: ***Turbo-Access***, a powerful implementation of the state-of-the-art B+tree ISAM technique; ***Turbo-Sort***, a super efficient implementation of the fastest data sorting algorithm, "Quicksort on disk". And much more.

Jerry Pournelle, BYTE: "The tools include a B+tree search and a sorting system; I've seen stuff like this, but not as well thought out, sell for hundreds of dollars."

Get started right away: free database! Included on every Toolbox disk is the source code to a working data base which demonstrates how powerful and easy to use the Turbo-Access system really is. Modify it to suit your individual needs or just compile it and run.

Remember, no royalties!

From Start to Finish in 300 pages. Turbo Tutor is for everyone, from novice to expert. Even if you've never programmed before, Turbo Tutor will get you started right away. If you already have some experience with Pascal or another programming language, Turbo Tutor will take you step by step through topics like data structures and pointers. If you're an expert, you'll love the sections detailing subjects such as "how to use assembly language routines with your Turbo Pascal programs."



A must. You'll find the source code for all the examples in the book on the accompanying disk ready to compile. Turbo Tutor might be the only reference on Pascal and programming you'll ever need.

$34.95

NOT COPY-PROTECTED

TURBO PASCAL FAMILY

Available at better dealers nationwide. Call (800) 556-2283 for the dealer nearest you. To order by Credit Card call (800) 255-8008, CA (800) 742-1133

Pascal 3.0	$ 69.95	____
Pascal w/8087	$109.90	____
Pascal w/BCD	$109.90	____
Pascal w/8087 & BCD	$124.95	____
Turbo Toolbox	$ 54.95	____
Turbo Graphix	$ 54.95	____
Turbo Tutor	$ 34.95	____

•These prices include shipping to all U.S. cities. All foreign orders add $10 per product ordered.

Carefully Describe your Computer System!

Mine is: ___ 8 bit ___ 16 bit
I Use: ___ PC-DOS ___ MS-DOS
___ CP/M 80 ___ CP/M 86

My computers' name/model is: ____________

The disk size I use is:
☐ 3½" ☐ 5¼" ☐ 8"

Name: ____________
Shipping Address: ____________
____
City: ____________
State: ________ Zip: ________
Telephone: ____________

Amount: (CA 6% tax) ____________
Payment: VISA MC BankDraft Check
Credit Card Expir. Date: ______/______
Card #: |_|_|_|_|_|_|_|_|_|
|_|_|_|_|_|_|_|_|_|

COD's and Purchase Orders WILL NOT be accepted by Borland. California residents: add 6% sales tax. Outside USA: add $10 and make payment by bank draft, payable in US dollars drawn on a US bank.

F20

BORLAND INTERNATIONAL

Software's Newest Direction
4585 Scotts Valley Drive
Scotts Valley, CA 95066
TELEX 172373

CIRCLE 9 ON READER SERVICE CARD