COMPUTER LANGUAGE™

$2.95
Canada $3.95

VOLUME 2, NUMBER 4

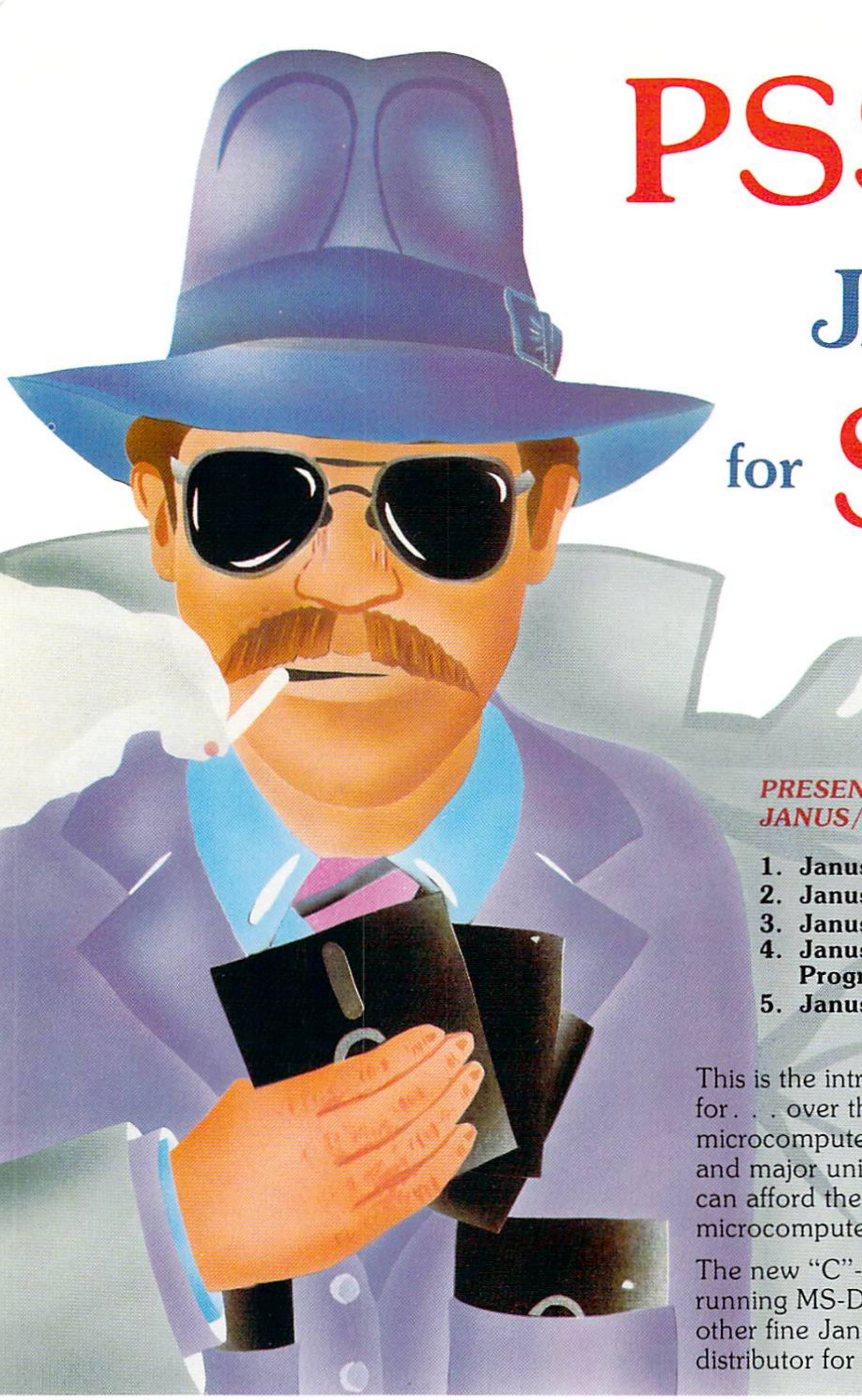
APRIL 1985

DISPERSION SORTING

FIVE MACINTOSH C COMPILERS COMPARED

MICROCOMPUTER COBOL COMPILER ANALYSIS

RECURSIVE PROCEDURES



PSSST... JANUS/Ada for $99.95!!

PRESENTING THE NEW JANUS/Ada C-PAK!!

1. **Janus/Ada Compiler**
2. **Janus/Ada Linker**
3. **Janus/Ada Libraries**
4. **Janus/Ada Example/ Programs**
5. **Janus/Ada User Manual**

AND THESE ADDED FEATURES!!

1. **Free User's Group**
2. ***$99.95 Discount* on the Janus/Ada D-Pak**
3. **No License!!**
4. **No Copy Protection!!!**
5. **Customer tested for over 3 years!!!**

This is the introductory Ada™ package you've been waiting for . . . over three years of actual field use, specifically on microcomputers, by the government, Fortune 500 businesses and major universities. Realistically priced, at $99.95, so *you* can afford the most popular Ada implementation used on microcomputers!

The new "C"-Pak is available for most microcomputers running MS-DOS, including the IBM PC AT™, as are all the other fine Janus/Ada programs. Call us or an authorized distributor for your copy today!

National Distributors

Westico, Inc.
25 Van Zant St.
Norwalk, CT 06855
(203) 853-6880

ASH II
7407 Marisol
Houston, TX 77083
(713) 933-1828

A.O.K. Computers
816 Easley St., Suite 615
Silver Springs, MD 20910
(301) 588-8446

Trinity Solutions
5340 Thornwood Dr., Suite 102
San Jose, CA 95123
(408) 226-0170

MicroProgramming, Inc.
P.O. Box 3356
Chatsworth, CA 91313
(818) 993-6475

International Distributors

Ada Australia
218 Lutwyche Rd.
Windsor 4030
QLD. Australia
(07) 57 9997

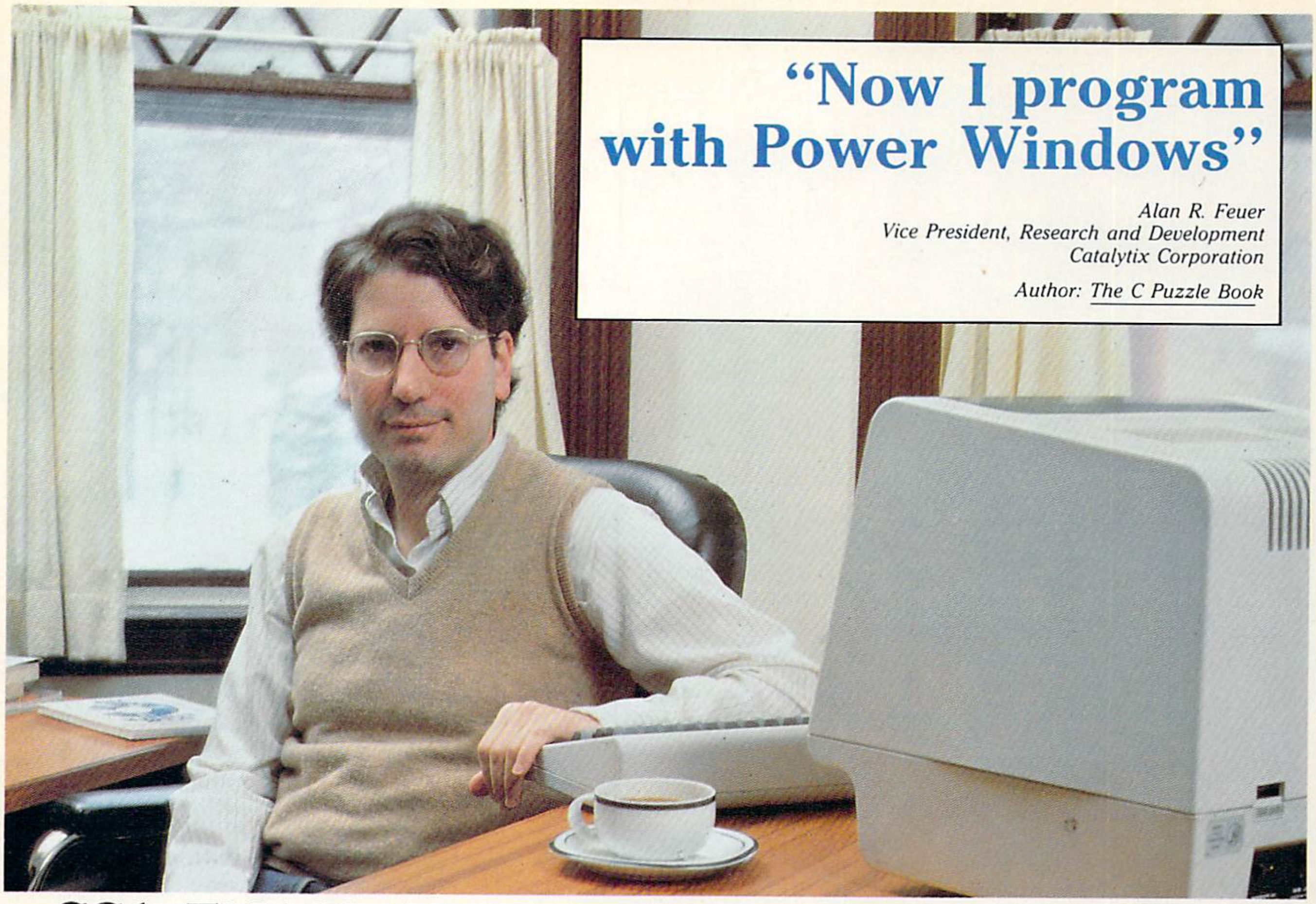
Progesco
155, rue du Faubourg
St. -Denis
75010 Paris
France
(1) 205.39. 47

Lifeboat, Inc. Japan
3-6, Kando-Nishikicho
Chiyoda-ku
Tokyo 101, JAPAN
03-293-4711

RR SOFTWARE, INC. *specialists in state of the art programming*

P.O. Box 1512 Madison, Wisconsin 53701
(608) 244-6436 TELEX 4998168

CIRCLE 58 ON READER SERVICE CARD



"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 82 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.

CL285

Imagine dBASE III™ running up to 20 times faster.

The time for Clipper has arrived.

Clipper introduces you to the time of your life.

Time is your most valuable commodity. Because how you spend your time, is how you live your life.

At Nantucket, we believe you should live life to the fullest.

Clipper, the first true compiler for dBASE III,™ is a timely example. Now, dBASE compiled by Clipper runs 2 to 20 times faster than dBASE with its standard interpreter.
A dBASE interpreter painstakingly checks and executes your source code one line at a time, every time you run a program. With Clipper, once you've debugged your source code, it's compiled into more efficient machine code. Your program runs without the time-consuming overhead of redundant translation. Clipper compiles all your existing and future dBASE III programs.

Developing a compiler for dBASE III was just a matter of time. Call your dealer or our toll free 800 number and ask for Clipper.

Then go make the most of your life time.

Nantucket

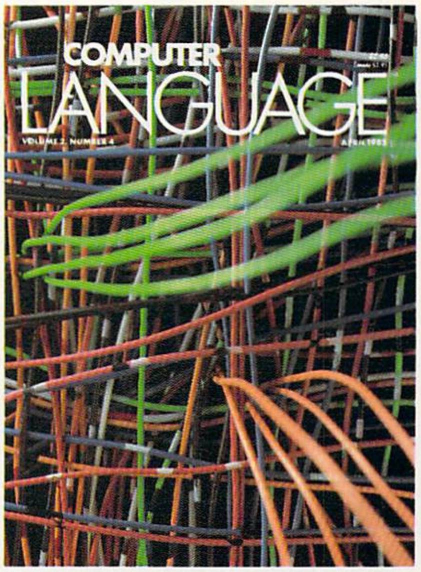
20456 Pacific Coast Hwy., Malibu, Ca. 90265 (800) 556-1234 ext. 225. In California (800) 441-2345 ext. 225

CIRCLE 59 ON READER SERVICE CARD

dBASE III is a registered trademark of Ashton-Tate

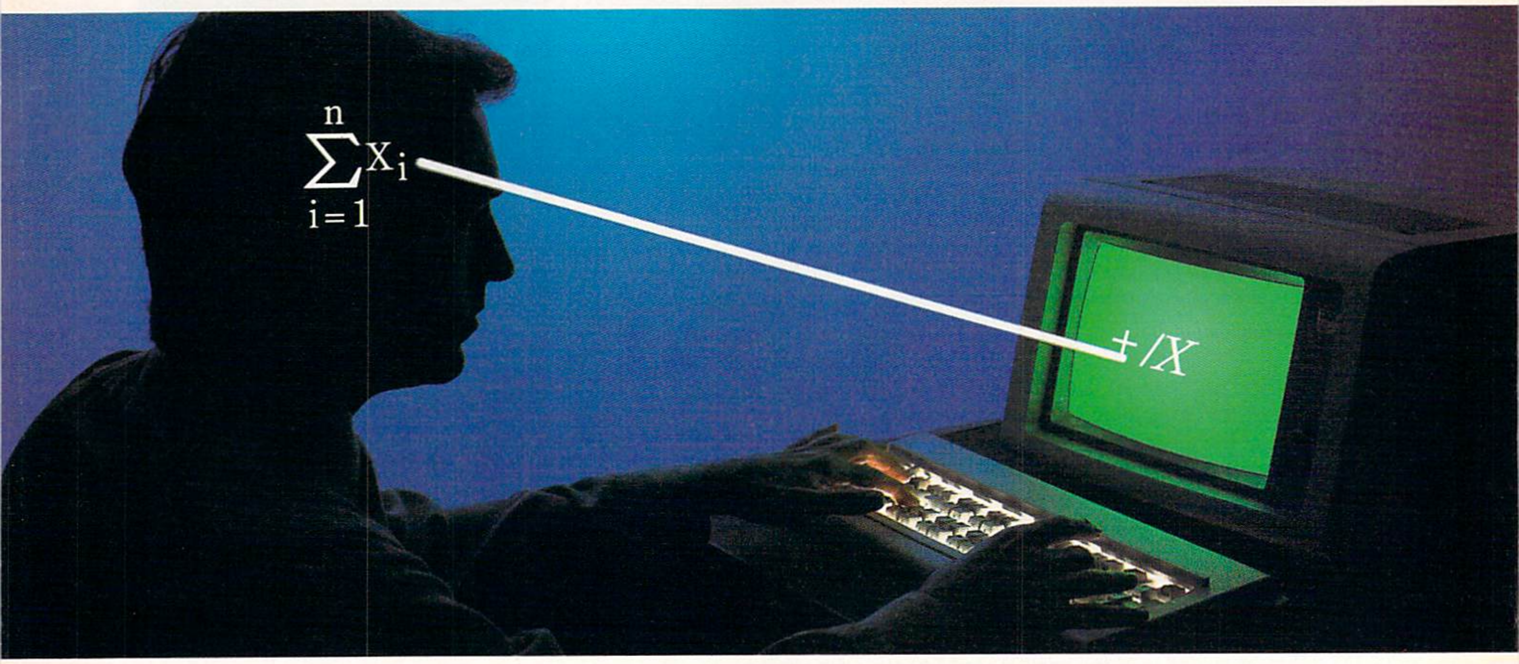
COMPUTER LANGUAGE

ARTICLES



DEPARTMENTS

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.

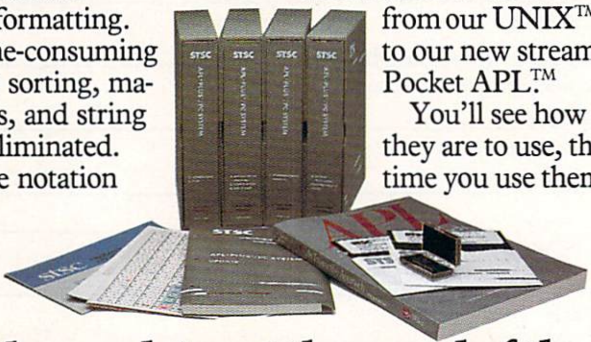
Writing time-consuming programs like sorting, matrix inversions, and string searching is eliminated. APL's concise notation already provides these. . .and more.

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 74 ON READER SERVICE CARD

Editor's Notes

Exciting news! Next month, we will present a very special feature in the pages of *COMPUTER LANGUAGE* . . .

Our debate columnist, Ken Takara, recently had the opportunity to interview perhaps the two most legendary figures in programming today—Donald Knuth and Niklaus Wirth.

Discussing a wide variety of subjects like programming style, language prejudices, art vs. science, and many other philosophical subjects, Knuth and Wirth review the past, present, and future of programming from their special perspectives.

In addition, Donald Knuth plays an indirect role in this month's *COMPUTER LANGUAGE*. A vital source for two features that appear in this issue is the well-known *Sorting and Searching* volume III of *The Art of Computer Programming*. This month we have two articles that explore sorting and searching from new angles. In "Sorting by Dispersion," author David Keil explores a sorting methodology that has really only been used for mainframe computations in the past. It has rarely been applied on microcomputers because of the inherent complexity of the algorithm's design, the higher memory overhead required to use it, and the marginal speed advantage at array sizes below a thousand.

But as hard disks and RAM chips of greater capacity come into use, it is possible that dispersion-based sorting algorithms may replace the more traditional sorting algorithms which have been based on comparison.

Hash coding is also treated from a different perspective by our CrossXthoughts columnist, Namir Clement Shammas, as he explores a searching method that involves data sorted in memory and on stored files. He shows how hashing techniques are important for data base programs as well as language compilers, interpreters, preprocessors, and any software that must search in a list of names quickly and efficiently.

On the electronic *COMPUTER LANGUAGE* side of things, another special news item this month is that our CompuServe data base has just passed the 10,000 caller mark! The amazing success we've had with this electronic medium, and with our two bulletin boards, has been almost as exciting as the magazine itself.

One notable change was also made in our CompuServe SIG—we have a new sysop. Jim Kyle, from Oklahoma City, Okla., will now be in charge of our *COMPUTER LANGUAGE* reader meeting place. He has plans for future real-time conferences with language founders and other notable technical people in our industry.

Just starting on our SIG are also various special interest groups—like the new language development club that Namir Clement Shammas has started (see the CrossXthoughts column, page p. 13) and special C and Forth language forums.

Finally, if you're a loyal C programmer or a person just interested in getting into C, don't forget to mark Sept. 16—18 on your calendars and meet us in Cambridge, Mass., for our C Seminar (see the advertisement on page 94). We will soon be announcing the C technical experts that will be speaking and conducting workshops for this special event. Maybe your company can pay for the trip?

[illegible]

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.

For those readers without access to a modem who desire a copy of program listings referred to but not printed in an issue, send $5 to *COMPUTER LANGUAGE*, attention: Listings Dept., 131 Townsend St., San Francisco, Calif. 94107. We will mail you a copy of all the listings not printed in this issue.

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.

COMPUTER LANGUAGE

EDITOR
Craig LaGrow

MANAGING EDITOR
Regina Starr Ridley

TECHNICAL EDITOR
John Halamka

PRODUCT REVIEW EDITOR
Hugh Byrne

EDITORIAL ASSISTANTS
Lorilee Biernacki, John Harrington

CONTRIBUTING EDITORS
Doug Millison, Tim Parker, Peter Reece, Namir Clement Shammas

INDUSTRY NEWS CONSULTANT
Bruce Lynch

SPECIAL PROJECTS MANAGER
Jan Dente

OPERATIONS CONSULTANT
Beatrice C. Blatteis

CIRCULATION COORDINATOR
Renato Sunico

ART DIRECTOR
Jeanne Schacht

COVER PHOTO
Dow Clement Photography

PRODUCTION ARTIST
Anne Doering

PRODUCTION
Barbara Luck, Steve Campbell, Kyle Houbolt

TECHNICAL CONSULTANT
Addison Sims

MARKETING CONSULTANT
Steve Rank

ACCOUNTING MANAGER
Lauren Kalkstein

PUBLISHER
Carl Landau

COMPUTER LANGUAGE *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 or Jan Dente 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 rate is pending at San Francisco, CA and additional mailing offices.

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.*

Programmers' Pfantasies by Phoenix.

Phoenix makes programmers' dreams come true. With the best-engineered, highest performance programming tools you can find. A full line of MS™-DOS/PC DOS programs and utilities no other company offers. All designed to help you write, test and deliver the best programs possible. Top-of-the-line quality at a price you can afford.

Finally, A Lint For MS-DOS.

Now you can get the full range of features C programmers working in UNIX™ have come to expect from their Lint program analyzer.

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. In fact, Pre-C outlints Lint, since you can handle analyses incrementally.

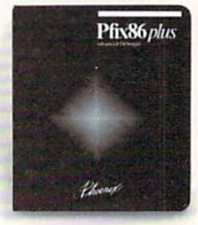
And, Pre-C's flexible library approach lets you maintain continuity across all the programs in your shop, whether you use Pre-C's pre-built libraries, pre-existing functions you already have, or some you might want to buy yourself.

Plus, you're not limited to one particular library, and Pre-C keeps track of all the libraries you're using to make sure that code correctly calls them. **$395.**

Assemble Programs Twice As Fast.

Pasm™86 assembles Masm files 2 to 3 times faster than Masm 3.0. Pasm86 supports 8086/88, 8087, 80186 and 80286 processors.

With Pasm86's built-in defaults, you can write code quickly since you won't spend hours learning all the control statements needed at the beginning of your program. You can define symbols on the command line. Decide whether you want error messages or not. And, put local symbols within procedures. **$295.**

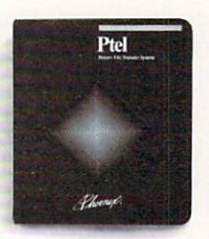


Still Fixing Bugs The Hard Way?

Pfix™86 Plus, the most advanced symbolic debugger on the market, eliminates the endless error searches through piles of listings. Locate instructions and data by symbolic name, using symbolic addresses. Handle larger, overlayed programs with ease.

An adjustable multiple-window display shows source and object code and data, breakpoint settings, current machine register and stack contents simultaneously. An in-line assembler allows program corrections directly in assembly language. Powerful breakpoint features run a program full speed until a loop has been performed n times.

With a single keystroke you can trace an instruction and the action will be immediately reflected in source, object, data, stack, and register windows. Another key begins a special trace mode that executes call and loop instructions at full speed. Designed to work with both Plink86 and MS™ LINK linkage editors. **$395.**



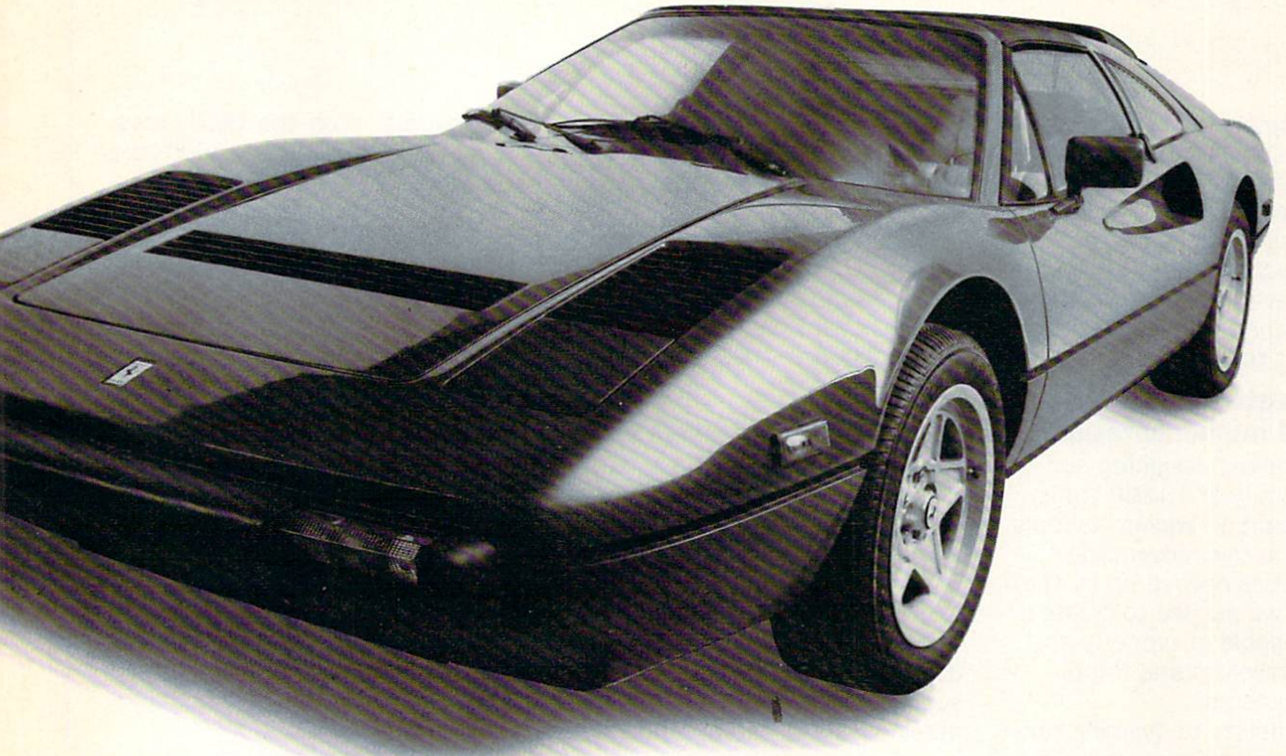
Get The Lead Out Of Binary File Transfer.

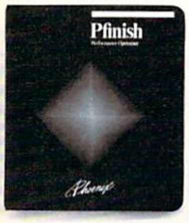
Ptel™ is the universal binary file transfer program for MS-DOS 2.0 or 3.0. You can move binary files fast and accurately. Upload or download groups of files from Bulletin Boards or remote computers. Move files between dissimilar machines and operating systems.

Ptel's advanced binary protocol, Telink, offers better-than-Modem7 accuracy and performance. Faster transfer speeds. An on-screen update of error correction, blocks, transferred, and time to complete.

Includes popular Modem7 and XModem protocols. With Checksum or CRC. Plus Kermit and ASCII. **$195.**

Special Thanks to Gasten Audrey of Framingham, MA.

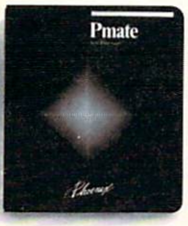




Maximize Your Program's Efficiency.

Pfinish™ delivers the fastest running programs possible. This performance analyzer lets you "zoom in" on the inefficient parts of your program. Whether written in assembly language. C. Pascal. Fortran. Even Basic. Unlike profilers available today, Pfinish understands the structure of your program and reports the amount of activity and time spent in its subroutines or functional groups. Pfinish analyzes both overlaid and memory resident programs. Down to the instruction level. Reports are displayed. Stored on disk. Or printed out. In tabular form or histograms.

Do a dynamic program scan. Identify the most frequently executed subroutines. Find inefficient code that costs your program valuable time. Rank subroutines by execution frequency. **$395.**



Why Work With A Primitive Editor?

More than a powerful editor, Pmate™ is a text processing language. An emulator of other editors. A language-specific editor for C, Pascal, and Fortran. Pmate™ can even run in the background!

You get full-screen, single-key editing. Ten editing buffers. Horizontal and vertical scrolling. A "garbage stack" buffer. A built-in macro language with variables, control statements, radix conversion, tracing and 120 commands that you can group and execute with a single keystroke. **$225.**

Why Squeeze Your Program More Than You Have To?

The Plink™86 overlay linkage editor brings modular programming to 8086/88-based micros. Write large and complex programs without worrying about memory constraints. Work on modules individually, link them into executable files. Use the same module in different programs. Changes the overlay structure of an existing program without recompiling. Use one overlay to access code and data in other overlays.

Plink86 links Intel-format modules. **$395.**

Call (1) 800-344-7200. In Massachusetts (617) 762-5030. Or, write.

Phoenix

Phoenix Computer Products Corp.
1420 Providence Highway Suite 115
Norwood, MA 02062

Pmate, Plink86, Pfix86 Plus, Pasm86, and Ptel are trademarks of Phoenix Software Associates Ltd. Pre-C and Pfinish are trademarks of Phoenix Computer Products Corporation. MS-DOS, and MS LINK are trademarks of Microsoft Corporation. UNIX is a trademark of AT&T Bell Laboratories.

CIRCLE 66 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.

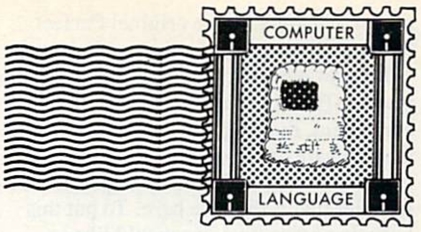
For quick response, call us at (303) 630-4000 or write:
Occam, P.O. Box 16000,
Colorado Springs, CO 80935.

inmos

inmos, and occam are trademarks of the Inmos Group of Companies

CIRCLE 40 ON READER SERVICE CARD

FEEDBACK



Touchy subjects

Dear Editor:

I was attracted to your February issue by the crisp cover photo and was delighted to see it was good food for thought as well. Your comparison of C compilers was courageous and I'm grateful. Criticizing compilers is about as welcome as a frank appraisal of your mate. Sometimes you have to live with the product to appreciate it.

Mike Pasini
San Francisco, Calif.

Waiting for Mac Cs

Dear Editor:

I really enjoyed the February issue on C. However, I am a Mac owner and programmer, and I was aghast you did not review the versions of C for the Mac. I am currently using Softworks C, which is a version of Whitesmiths' C compiler. Please review the Mac C compilers so that I might get some insight into the others.

Kyle Jedrusiak
Hazlet, N.J.

Five Macintosh C compilers are reviewed in this issue, beginning on page 83.—Ed.

Toolworks C/80 rebuttal

Dear Editor:

We at The Software Toolworks are puzzled and distressed at the inaccurate evaluation of the Toolworks C/80 compiler for CP/M (*COMPUTER LANGUAGE*, February 1985). *COMPUTER LANGUAGE*'s panel of experts made many factual errors which we will attempt to correct.

Toolworks C/80 is not "a late entry into the group of CP/M compilers," and it does not cost "about $90." It sells for $79.90, including the integer compiler ($49.95) plus the MATHPAK ($29.95), which adds true *float* and *long* data types. It has been on the market since 1980.

At this price, you would not expect a full Kernighan and Ritchie (K&R) C compiler. Indeed, of the six CP/M C compilers reviewed, only one or two (depending on where in the review you look) are full K&R C. Toolworks C/80 is not. The features we do not support are double-precision arithmetic, bit fields, *typedefs*, arguments to macros, #line, and declarations within nested blocks.

We were shocked, then, that not once, but twice, *COMPUTER LANGUAGE* incorrectly stated that we advertise a full C and then took us to task for not meeting the claim. The Software Toolworks has a fine reputation for quality software, bargain prices, and square dealing with customers, built up over five years. Since we do not spend a great deal of money advertising our products (a fact reflected in our low prices), our reputation is particularly dependent on objective reviews. This inaccurate allegation of misleading advertising is libelous and very damaging to us, and we hope that *COMPUTER LANGUAGE* will see fit to formally retract it.

Perhaps the panel of review experts did not read our advertising carefully. We do not use the terms "full C" or "full K&R C". We do use the phrase "full-featured," which doesn't mean the same thing at all. We use it to refer to such features as built-in command line wildcard expansion and I/O redirection, ROMable code, Macro-80 and RMAC compatibility, execution time profiling, in-line assembler code, and an extensive library. These are important features, and we think *COMPUTER LANGUAGE*'s failure to mention any of them contributes to the inaccurate impression given by the review.

We acknowledge that this last complaint goes beyond correcting factual errors and takes issue with the reviewer's personal opinion and choice of what points to make in limited space. Still, we do feel that C/80 was presented more negatively than other compilers. For example, Q/C was praised for how much of K&R it supports for under $100, yet C/80, which supports far more of the language for less money, was attacked for its points of difference with K&R. The reviewer also liked Q/C's run-time trace and support for several assemblers. C/80 has similar features, but the only mention of either was a backhanded acknowledgement of "rudimentary verification tools."

Toolworks C/80 failed to compile the Deref benchmark because it is limited to seven levels of indirection in a type declaration. We don't feel that this is even a mild drawback.

The review said that the compiler "exploded on Deref." We called Jim Reed, who actually ran the benchmark, and he was very kind in helping us determine what actually happened. The full Deref benchmark produces an error message; the explanation of the message in the manual points out the seven-level limit. When Deref was reduced to seven levels, the program compiled without error. When run, it printed out garbage and returned to the A prompt. The statement that the compiler failed was simply bad English writing on the part of the reviewer, who meant to say that the compiled program failed.

But the problem was in the benchmark, not in the compiler. The program contains the statement:

```
printf("%u loops\n",LOOPS)
```

where LOOPS is *#define*d as 50000. Since 50000 exceeds the *signed* integer size limit, it is compiled as a *long* constant—consistent with K&R—and should be printed with a *%1d* conversion, not *%d*. Because of the way *printf* is implemented on most compilers, the program, while buggy, will still run, but on Toolworks C/80, it prints garbage. This is not a fault of the compiler; if the arguments and the conversions do not agree, nonsense may result, again according to K&R. Furthermore, the program does not explode or lose control, as the reviewers thought; it runs correctly but the printout is wrong.

Our experience also contradicts the reviewers' hesitancy to use Toolworks C/80 for "writing a large program." We have used Toolworks C/80 to compile the compiler itself, our MyCalc "full-featured" (but not full K&R) spreadsheet, and many other large programs. Others have used Toolworks C/80 to implement products like the Random House Proof-

Illustration: Anne Doering

reader and much of the original Perfect Software line.

It is always difficult to reply to a critical review. The reviewers are presumed to be competent, thorough, and objective, and the software developer appears to be a crybaby. We trust the reader to judge whether that is the case here. To put this letter in perspective, we would like to point out that The Software Toolworks has never before, in five years of publishing more that 40 software products, objected in this strong fashion to a review.

Walt Bilofsky, chairman
The Software Toolworks
Sherman Oaks, Calif.

A pat on the back

Dear Editor:

We were very pleased to see the survey of C compilers in the February issue of *COMPUTER LANGUAGE*. We thought the survey as a whole was pleasant, informative reading, and correctly represented the products with which we are directly familiar. It is important because it has been well over a year since such a wide ranging view has been taken (since *BYTE*'s September 1983 issue).

One important improvement for future surveys when using the style of saying at least one bad thing and one good thing about each product would be to add ranks or a ranked summary for the products. Readers have to do a lot of reading between the lines to separate the losers from the superior products.

Also the Deref benchmark seems technically irrelevant. Not only are such constructions extremely unlikely to appear in practice, but the ANSI C standard will only require that six levels of indirection be supported by a conforming implementation. At least two other fully competent C implementations, from Wizard and Computer Innovations, would appear upon casual reading of Table 2 to be defective in some major way. Why have a benchmark that shows half of the products won't process language features that never occur in real programs?

In closing, the look and the reading of this issue make it the best yet: congratulations and thanks.

Terry Colligan, president
Ben Williams, vice president
Rational Systems Inc.
Natick, Mass.

FORTRAN's many virtues

Dear Editor:

Anthony Skjellum's article "C Instead of FORTRAN?" (*COMPUTER LANGUAGE*, February 1985) gave a nice tutorial on C for numerical applications but could hardly be called a comparison of the languages.

For scientific and engineering problems, FORTRAN has the following virtues—principally numerical and very often concerned with linear (matrix) algebra—over C:

■ Multiple dimension matrices can be passed and handled more naturally. It is legal in FORTRAN to say:

```
SUBROUTINE F(A,N,M)
DIMENSION A(N,M)
```

whereas C has nothing similar. This facilitates writing library routines in which the dimensions are passed as arguments.

■ The FORTRAN default convention of starting indices with 1 is more natural than the C convention of zero-based arrays. Most FORTRANs allow this to be changed but not C. In FORTRAN, if I define:

```
PARAMETER(N=20)
DIMENSION A(N)
```

then the last element in *A* is *A(N)*, loops run from 1 to *N*, etc. In C, for the same array:

```
#define N 20
double a[N-1];
```

is needed; the possibilities for confusion and error abound.

■ FORTRAN has the *COMPLEX* type. For many electrical engineering applications, complex numbers are a godsend.

■ C generally does all its floating arithmetic as double. This can be wasteful of time. FORTRAN allows me to decide what needs double precision.

■ For exponentiation in C, I must use *pow(x,y)*, with the argument floats. In FORTRAN, *A**Y* will work with *Y* an integer, for *A* real or integer, as well. Often this is optimized, that is, if *Y*=2, a multiplication is done instead of finding logarithms, multiplying, and exponentiation.

■ The library of functions, including inline functions, is much more extensive for numerical applications in FORTRAN. I can write:

```
BIG= AMAX1 ( A,B,C,D)
```

where *AMAX1* can have any number or arguments, as it is expanded in-line. C has no similar capability.

■ Finally, there is a heck of a lot of FORTRAN source out there for most any conceivable application. Aside from *EISPAK*, *LINPAK*, and *BLAS* (basic linear algebra subroutines), many books on numerical analysis have excellent routines in FORTRAN (for example, Forsythe, Malcom, and Moler's *Mathematics of Computation* has routines for singular value decomposition, solving linear systems, integration, systems of differential equations, finding roots, and optimization).

What advantages does C have? Mostly in the non-numeric applications, for example, interfaces to the operating system. I often use FORTRAN to find the numerical results and plot them with C (Microsoft FORTRAN on a Corona PC, with a program called GraphiC using DeSmet C. Microsoft FORTRAN for MS-DOS machines has all of the features mentioned, such as type *COMPLEX* and *PARAMETER*). The best of all worlds would be interfaces between the two (I have written one between Aztec C and Microsoft FORTRAN for CP/M).

The best language still depends on the application.

Louis Baker
Albuquerque, N.M.

C—no serious challenger

Dear Editor:

I cannot stand back any longer. I must take exception to the C lovers who appear to be inhabiting computer magazines. I take particular offense to Anthony Skjellum's article "C Instead of FORTRAN?" I will go through some of the arguments raised by the author and provide some counterarguments.

"FORTRAN did not provide a programming environment that was conducive to structured, modular programming." Bunk! A prime example of structured modular programming is the IMSL and LINPACK libraries that have been developed. With only a few lines of code and these libraries (standard across micros, minis, and mainframes alike) one can do virtually any mathematical operation one wishes to perform.

Skjellum states that the lack of a universal FORTRAN standard forces many users to fall back on FORTRAN 66. It is interesting to note that in the same issue of the magazine—in fact, the previous article—is a discussion of the beginning of a C standard. FORTRAN has at least the 66 and usually the FORTRAN 77 standard.

"C . . . is widely available with essentially no dialects." Again I must beg to differ. Other articles in the same issue serve to prove my point. In the article on standardizing C are several examples of compiler-dependent differences. Then in the software reviews at the end of the issue is more evidence. Look at the tables and note the number of blanks. In addition, read the reviewers' comments regarding the problems of compiling the benchmarks on all of those different, but "no dialects" compilers.

If C is supposed to be taken seriously as a scientific and engineering language then further information will need to be provided in software reviews, preferably in the tables as well as the text. Information is needed on the support or lack of same for numeric coprocessors, if they exist. In the case of compilers designed to use Intel microprocessors, we need to know about the memory models supported. Finally, what about support for complex variables?

While C is a highly structured language with all sorts of silly symbols, I cannot believe that it is a serious challenger to FORTRAN. Who is going to go to the expense of converting those large libraries to a language that is not even close to a standard? Let C stay in the domain where it is most useful and let FORTRAN take care of the math, perhaps even going as far as having subroutines in many specialized languages. I think the author made this point in his second paragraph, but he then proceeded to ignore it. Use the language best suited to the application. Period.

David W. Hopper, P.Eng.
Toronto, Ont.

Worth the wait

Dear Editor:

Namir Shammas's CrossXthoughts column in the February *COMPUTER LANGUAGE* was thought provoking. The column really sounds exciting.

The level of syntax checking put into the program NILE is just right. You don't want to get carried away with rules or invent a new programming language. It's just a means of human communication that can be improved somewhat with the aid of a computer, in much the same way you might use a spelling or style checker on your prose.

The thing I like most about NILE, though, is the list of undefined procedures and functions it provides. Using that, you can be constantly aware of how much work still lies ahead of you. A lot of my time has been spent perusing routines I've already designed, looking for what I have to do next. This will help.

The angle brackets enclosing the keywords were a problem. Notice how neat and clean Listing 5 looks as it is printed in your column. Compare it, please, with the appearance of the input to NILE. Brian Kernighan and P.J. Plauger, although they were addressing a slightly different subject, made an appropriate comment in *Software Tools*: "For supplying arguments to programs, we feel that commas and parentheses as argument separators merely add noise and keystrokes and are better avoided." I urge you to reconsider this aspect of PPL.

There were two errors in the copy of NILE that I downloaded from the *COMPUTER LANGUAGE* Bulletin Board Service. In procedure *INITIALIZE*, the link on *OTHERWISE* points to itself. That is, the line was:

```
Save(Kword[13],'OTHERWISE',
    1,13,0,0,0);
```

It should be:

```
Save(Kword[13],'OTHERWISE',
    1,12,0,0,0);
```

Also, the *THEN* could not follow an *ELSEif*. That is, the line was:

```
Save (Kword[7],'THEN',1,6,0,0,0);
```

It should be:

```
Save(Kword[7],'THEN',2,6,9,0,0);
```

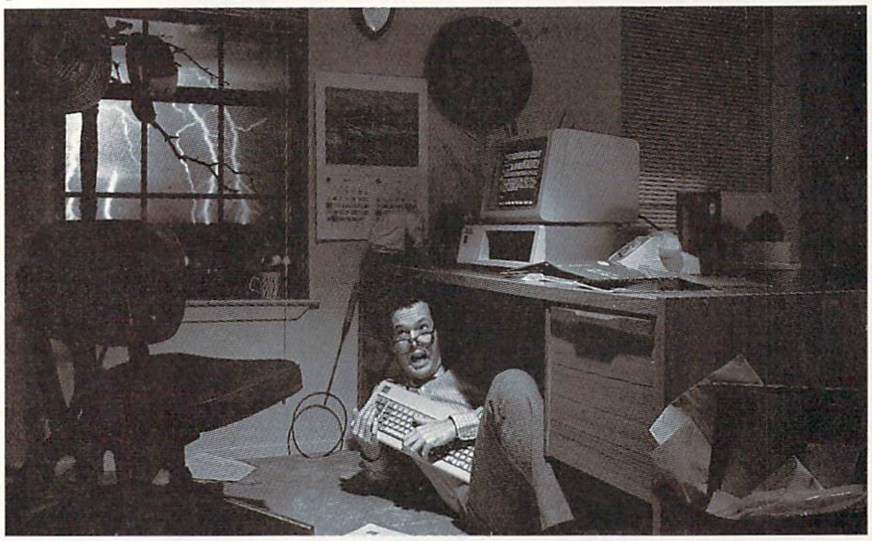
A question: whence the name NILE?

Bill Blum
Daly City, Calif.

Author Namir Clement Shammas responds: Thanks for noting the errors in NILE. An updated version is now on the BBS.

As for the name NILE—until recently I was doing the business under the company name of Pyramid Software, so the name NILE seemed an appropriate choice.

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 75 ON READER SERVICE CARD

Don't cripple Pascal

Dear Editor:

In Feedback in the January 1985 issue of *COMPUTER LANGUAGE*, Starret C. Kennedy complains that Pascal suffers from something to which he/she refers as the "triad syndrome." Kennedy apparently feels that Pascal is not a good teaching language (this is the impression one gets, though Kennedy never really addresses the issue directly) due to this and some undefined problem with the looping constructs.

I believe Pascal to be an excellent teaching language, and I consider Pascal the language of choice for some programming tasks. I do not consider Pascal to be the do all and end all of programming languages, nor do I believe that there shall ever be such a language.

Kennedy complains that Pascal supports three assignment operators (:=, :, and =). Balderdash and hogwash. Kennedy has obviously missed the point. There is but one "assignment" operator in Pascal, the := symbol. This symbol is used to denote assignment of the results of an expression on the right side of an operator to the location identified by the value of the expression on the left side of an operator.

The : symbol is not by any stretch of the imagination an assignment operator. Rather, the : is a binding operator, binding a type attribute to an identifier (or group of identifiers in a declaration. A declaration is a statement to the compiler that requests storage space for an identifier and binds certain attributes to that identifier.

The = symbol, in the context in which Kennedy refers to it, is a definition operator. This symbol defines for the compiler an identifier with a constant value. This value is then replaced by text substitution for the name of the identifier whenever that identifier is encountered in the source file during compilation.

Assignment is a run-time phenomenon—it does not take place at compile time. Declaration and definition are compile-time operations and likewise do not occur at run time. These distinctions may appear to be nitpicky but, in fact, they are essential to the understanding of compiler operation and utilization.

Yes, I believe that unnecessary complexity in a programming language is undesirable. Yes, I concur that user-friendly software is essential in the computing industry. However, programming languages are not applications programs and are not intended for the neophyte who merely wishes to get the payroll out, the inventory counted, and the daily tally sheets to bed.

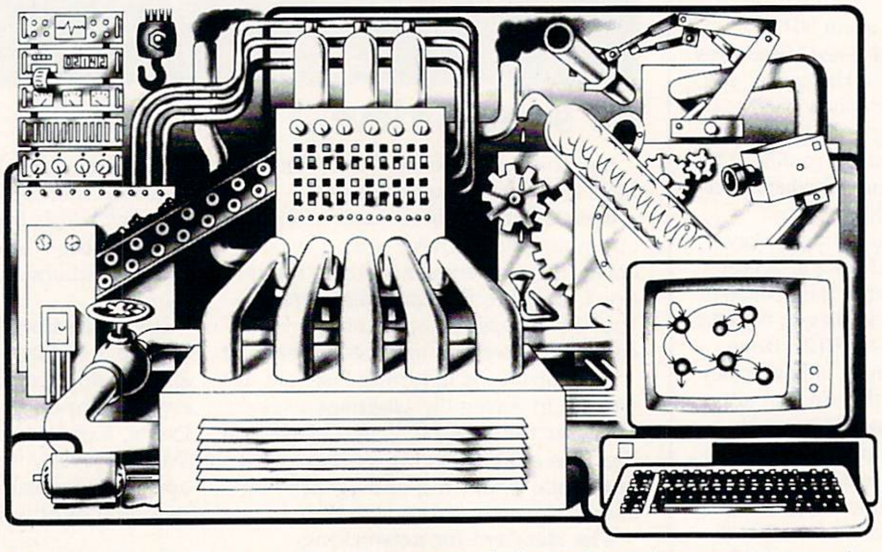
Persons who are learning to program need to be prepared to learn a complex, detailed and highly sophisticated methodology for causing the computer to accomplish specifically defined and carefully formulated tasks. Overgeneralization of control flow constructs and operator conventions will not, as implied by Kennedy, benefit the programmer. They will simply make the task of designing, implementing and analyzing code more difficult, more tedious, and less efficient.

I am a strong proponent of computer literacy and would like to see as many people as possible learn to perform at least simple programming. I do not believe that a language that is specifically designed as a teaching language for computer sciences ought to be crippled by oversimplification to cater to the lack of technical expertise and understanding of the general public. Pascal is such a language and, as such, is a success.

Thomas Keller
Santa Rosa, Calif.

Illustration: Anne Doering

Hard problems? **Csharp** can help!



Cut your development time with C source code for realtime data acquisition and control. The **Csharp Realtime Toolkit** includes: graphics, event handling, scheduling, and state systems. Processor, device, and operating system independent. Price: $600

SYSTEMS GUILD

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

CIRCLE 89 ON READER SERVICE CARD

Search by hashing

By Namir Clement Shammas

In the last issue we talked about mathematical parsers and interpreters. This month, let's take a look at searching methods that use hashing techniques, employing data in memory as well as on stored files.

Hashing techniques are important for data base programs as well as language compilers, interpreters, preprocessors and any software that must search in a list of names quickly and efficiently.

The idea behind the hash-based search is to transform the search key, be it numeric or alphanumeric, into a unique address used to locate the information. This procedure offers a fast search capability to a computer system, useful, for example, when an airline needs to pull out passenger reservation information without delays. We will explore hashing, its merits and faults, and suggest some remedies for its weaknesses.

The technique in question essentially relies on the use of a hash function, *H(x)*, that maps the search key onto (we hope) a unique address in a predefined range of addresses.

The address range limitation is one weakness of hashing. The majority of hash functions generate random values (or addresses). Thus a list of ordered keys would create scattered addresses that do not reflect the original order, as in:

If key X > key Y . . . then
H(X) > H(Y)

is not always true.

This immediately tells us that using such hash functions deprive us of the ability to easily maintain sorted lists, which is the price to pay for speed. If sorting is required, one can use an order-preserving hashing function, such that:

If key X > key Y . . . then
H(X) > H(Y)

Since the hashing functions produce numeric addresses, textual keys must be converted into numerical values. Using the ASCII code is one way to go, however, each character yields two or three digits.

Shifting values can also be used. The addresses calculated can be used to either store the data records when one search key is used or their indices when the data is searched by multiple keys. The records would be stored sequentially in the latter case.

What are the types of hashing functions? How do they work? What are their strengths and weaknesses? These are the questions to answer to properly choose a hashing function.

To answer the first question, keep in mind that there are as many types of hash functions as the imagination can create. Here are a few:

Truncation. A specific set of digits or numbers, such as the second, third and fifth digits in an eight digit key is selected. The rest is ignored. This method provides a fast and easy way of obtaining an address in the hash table but often fails to provide even distribution in the table.

Folding. The original numeric key is divided into segments that are simply added. If the result is bigger than the hash table size, the address is taken as (sum of digits) modulus (table size). For example, a 10 digit key (1234567890) is folded into 34567 and 12890. Their sum yields 47457. The latter is a suitable address for a hash table with a size of 50,000. The method is reportedly very good at randomizing the key. However, it depends on the key sequence and is not very reliable.

Modular arithmetic. The address is obtained as the remainder from dividing the numeric key by the hash table size. This very popular method yields a good spread over the hash table.

Midsquare method. This method yields an address by squaring the numeric key and selecting an appropriate number of digits from the middle of the square. This method has been criticized by some but proven to be effective with certain types of keys.

Length dependent method. The length of the key is used along with some portion of the key to produce an address. On many occasions the address is seen as an

New language project!

In the March CrossXthoughts column I invited and challenged the readers of *COMPUTER LANGUAGE* to create a new programming language or modify an existing one. Since then, I've had a lot of interaction with readers, especially via CompuServe. The project received a lot of enthusiasm. The development of a new, general-purpose language will give us an opportunity to learn how a computer is designed and how it works.

This announcement includes an invitation for you to participate. A wide variety of talents are needed for the numerous aspects of the project. We will work as teams, each handling a specific task. There will be a lot of interaction between the teams. *COMPUTER LANGUAGE* will dedicate a regular sidebar identical to this one as space for important announcements and progress reports. The magazine will also help in mailing reports and material to those involved. The editor has agreed to publish the project's document and sell it at cost.

I believe that this can turn into one of the most dynamic and rewarding projects for all of us. Keep in mind that there are many basic and initial decisions to make. For example, what is the best way to communicate? Not everybody has a modem or a CompuServe subscription. (The old U.S. mail is perfectly fine with me!) Another decision concerns where the project documents will be stored electronically.

Let us use our talents to create a better programming environment. You are welcome to send me, in care of this magazine, your name, mailing address, phone number and area of interest in the project. For more information, write to: Namir Shammas, *COMPUTER LANGUAGE*, 131 Townsend St., San Francisco, Calif. 94107.

C

C Libraries

C Windows
Best You Can Get!

SIX C LIBRARIES
250 Fully Tested Functions
FUNCTIONS YOU DON'T HAVE BUT NEED!
All Source Code. No royalties.
More & Better Functions
Best Documentation

41 screen handling/graphic functions	$39.95
40 cursor/keyboard/data input functions	$39.95
69 superior string functions	$59.95
15 system status & control functions	$19.95
58 utility/DOS/BIOS/time/date functions	$49.95
30 printer control functions	$29.95

No Matter What Else You Have, Get These!
Any 3 Libraries $59.95-All 6 Libraries $99.95

C Windows
PROFESSIONAL WINDOW MANAGEMENT
Overlays,Borders,Popup Menus,
Help Windows,Status-Line,Color Highlighting And More!!!
C Windows: Complete Source Code.....$89.95

THE PROFILER
by DWB ASSOCIATES
The Cadillac of profilers..$125.00

COMBINATION OFFER
C WINDOWS PLUS 6 LIBRARIES
For $149.95
C WINDOWS PLUS THE PROFILER
Both For $169.95
SIX LIBRARIES & THE PROFILER
Both For $179.95
C WINDOWS & 6 C LIBRARIES
& THE PROFILERS
A $315. Value All For $219.95

Entelekon
SOFTWARE SYSTEMS

ENTELEKON 12118 KIMBERLEY
HOUSTON, TX. 77024 (713)-468-4412

CIRCLE 24 ON READER SERVICE CARD

intermediate value that yields the sought location via modulus calculations with the hash table size (as with modular arithmetic). This method is useful in dealing with alphanumeric keys.

Many of the books listed in the reference section discuss the types of hashing functions in more detail.

A reader may ask, "Is there any hashing function that guarantees no two different keys will yield the same address?" The answer is no.

Collision, as the effect is called, is almost certain and is caused by the choice of hashing function or the use of a small size for the hash table. Increasing the size will not eliminate collision completely, only decrease it.

The next question is, "What methods do we use to deal with collision?" The good news is that there are a good number. One category, called open addressing, includes:

Linear probing. This method is very simple. If the hashing function gives an address that turns out to be already occupied, it performs a sequential search for the locations that follow the collision site until an empty location is found. Keep in mind that the hash table should be regarded as a circular list. The major disadvantage of linear probing is the formation of clusters and an uneven distribution.

Quadratic probing. This technique is a modification of linear probing. If there is a collision at address $H(x)$, then start probing at addresses $H(x)+1$, $H(x)+4$, $H(x)+9$, and so on. While this method resolves the problem of clustering, you can clearly see that not all the locations of the hash table will be probed.

Key dependent probing. This method uses part of the search key to decide the magnitude of an address offset once collision occurs.

Increment functions. A more sophisticated approach is to use a set of hash functions instead of one function. If the address from the first one causes collision, then we apply the second hash function to calculate an address. If collision occurs with the second, we use the third hash function and so on.

Chaining, another method for dealing with collision, dictates the use of two types of storage locations: the hash table and the overflow area. When a collision occurs, say between the newly added key Y and resident key X, the former is stored in the overflow area at the next available location. Key X sets a pointer to the site where key Y is stored, forming a linked list. If another collision occurs between another inserted key, Z and X, then Z is also stored in the overflow area. To maintain the linked list, key Y will contain the pointer for the site of key Z.

The advantage of chaining lies in economical storage, which becomes more evident as the records stored are larger. The hash table need not be oversized.

The disadvantage of chaining becomes apparent when searching through an unsorted list. If a sought key is nonexistent, then the entire linked list is searched. However, chaining works better than open addressing techniques. *Data Structures and Program Design* uses mathematical proof to demonstrate this.

Imagine searching in memory, using the chaining method to resolve collision. An alternative to maintaining linked lists in the overflow area is to use binary trees, especially when a collision occurs frequently. This makes searching in the overflow area faster since the keys in the overflow area are now sorted. The price to pay is that maintaining binary trees is more involved, especially when it comes to deleting.

The idea of using binary trees becomes more appealing when order-preserving hash functions are used. They can simply use the first one or two most significant digits or letters in the search key.

The combination of hash table and binary trees—I'll call it the H-Tree—allows for fast searching and sorting in memory. The hash table will contain a set of roots for binary trees. By using the divide-and-conquer strategy, we are scanning fewer nodes at a time while adding, deleting and searching.

What about searching in files saved on disks? First, we must remember that access time of a mechanical device is much greater than that for memory. So we need to perform the least number of disk accesses.

Second, since I/O operations occur via memory blocks or buckets of 256 bytes or its multiples, we can collect a number of keys in a bucket. Thus we can store the data records separately and create another storage area for the pages containing the search keys.

Again we have two storage areas for keys: the hash table and overflow area. The mechanism is very similar to that of memory-based search. The difference is that single keys in a memory-based search are replaced by buckets. Thus the hash table will contain a number of buckets, each containing colliding keys. When a hash table bucket overflows, a bucket in the overflow area is created, and the two buckets link using a pointer.

To decrease the number of collisions, bigger hash tables are required. This is a disadvantage, because we are reserving a lot of space and assuming even distribution of the keys. The following strategy deals with the problem of excessive space.

Using a two-dimensional hash table and two distinct hash functions, the first hash function determines the row address, and the second determines the column address. Any keys colliding due to the first hash function address are separated by the second hash function. The overflow area is still used, for nothing is collision-proof.

So far we have handled the collision problem but still have to solve the space allocation problem, especially in dealing with buckets. Here is the way out: rather than creating the space for the key buckets, we instead create, in memory, a two-dimensional map of pointers. As for the key buckets, we store them up in chronological order as they are introduced in the system.

The map needs to be loaded at the beginning and stored at the end of a data-manipulation session. Here is how the scheme works:

Initialization. Assign the data file and the key-buckets file and zero all data counters and pointers. Initially the elements of the latter are pointing nowhere.

Start-up. As data starts to flow in the system, update the number of records and extract the search key. Use the latter to determine the row and column address in the key map. If the latter is zero, the creation of a new bucket is required. In any case the key is stored in a slot located inside a bucket. Figure 1 shows a fairly empty key map. The first record added has a key equal to 16. So *Map(1,6)* is the first-used pointer to the first bucket. A second record with a key equal to 36 is entered and is used to point to the second bucket, and so on.

Routine addition. As buckets are filled, new ones are created and pointers are used to maintain the linked list of buckets. Suppose that we add 19 more records, all with keys equal to 16. If each bucket holds 20 keys, then we have filled the first bucket. As we add the twenty-second record, always with a key equal to 16, then a new bucket is created (number three to be exact) and the key points to record number 22.

Deletion. It is usually more difficult to delete parts of data structures than to build them. As records and keys are deleted, they create gaps in the data lists. One can construct additional lists for the latter gaps. The lists are either used in periodic packing of files or to fill the gaps by storing newly added records and keys.

An alternative way to keep track of the lists is to move the last element in the linked lists of records and keys into the location of the deleted record and key. One must take into account the effect of emptying buckets. Their space must be regained. This will require that the buckets be double-linked lists to allow travel through the lists in both directions.

Listing 1 has a few procedures—written in programmer's pseudo listing (PPL)—to demonstrate the code for addition and search. I have not included the code for deletion due to space limitations.

If this technique is used with order-preserving hash functions then the ability to perform sorting is vastly improved. Once more the principle of divide-and-conquer is put to service. To sort, a program would go through the map in a systematic way (reflecting ascending or descending order) and read the unsorted list linked to that map element. The keys in each of the lists are then sorted in memory and the result output. Since there are relatively few elements in a set of linked lists, sorting should not be too painful.

The other alternative is to maintain the linked lists of buckets in a sorted order, which slows the process of adding and deleting data but yields an improved search.

Data structures and management are of interest to most, if not all, programmers. Sorting and searching play a vital role in data processing. It is probably the subject most talked about and most researched. It is also an art.

The numerous algorithms involved and their modification form a vast number of methods. It is indeed a fascinating subject. Let me hear from you. If you have some code you developed that performs

```
Mapped-keys strategy (pointers to the records belong to the first
keys added to a bucket)

    RAM key map

     0  1  2  3  4  5  6
   ---------------------
0  |  |  |  |  |  |  |  |              Key buckets      Data records
   ---------------------
1  |  |  |  |  |  |  |16|------------v
   ---------------------                            1
2  |  |  |  |  |  |  |  |          +--| 16  | ---------->|16 data    |
   ---------------------           |  +-----+        2   -----------
3  |  |  |  |  |  |  |36| -------->| 36  | ---------->|36 data    |
   ---------------------           |  +-----+        3   -----------
                                   ->| 16  |----          |          |
                                      +-----+    |        -----------
                                                 |        ...........
                                                 |   22   -----------
                                                 ------> |16 data    |
                                                          -----------
```

Figure 1.

fast searching, then here is the good news: you may be the winner in our searching minicontest!

We will award prizes (at least a free one-year subscription extension) for the best three well-written and fastest (and I mean fast) codes. Because I may not have the same hardware and language implementation, I will rely on your speed test results, which should be included. The winners will be announced in four months.

In the next issue we will talk more about external hashing. Some interesting techniques overcome the limitations of a predetermined hash table size. I leave you with a list of references on the subsets. ■

References

Hanson, O., 1982. *Design of Computer Data Files*. Rockville, Md.: Computer Science Press.

Horowitz, E. and Sahni, S., 1982. *Fundamentals of Data Structures*. Rockville, Md.: Computer Science Press.

Horowitz, E. and Sahni, S., 1984. *Fundamentals of Data Structures in Pascal*. Rockville, Md.: Computer Science Press.

Knuth, D., 1973. *The Art of Computer Programming, Vol. 3: Sorting and Searching*. Reading, Mass.: Addison-Wesley.

Kruse, R.L., 1984. *Data Structures & Program Design*. Englewood Cliffs, N.J.: Prentice-Hall.

Melhorn, K., 1984. *Sorting and Searching*. New York, N.Y.: Springer-Verlag.

Tenenbaum, A.M. and Augenstein, M.J., 1981. *Data Structures Using Pascal*. Englewood Cliffs, N.J.: Prentice-Hall.

Tremblay, J. and Sorenson, P.G., 1984. *An Introduction to Data Structures with Applications*. New York, N.Y.: McGraw-Hill.

Ullman, J.D., 1982. *Database Systems*, Second Edition. Rockville, Md.: Computer Science Press.

Weiderhold, G., 1983. *Database Design*, Second Edition. New York, N.Y.: McGraw-Hill.

Wirth, N., 1976. *Algorithms + Data Structures = Programs*. Englewood Cliffs, N.J.: Prentice-Hall.

```
A collection of procedures, written in PPL, for initializing,
adding, and searching for records

-- Data types will be defined in a Pascal-like style.

-- RECORD : Anytype;
-- KeyData : record Key : Anytype; RecordPointer : integer end;
-- Map : array[1..MAXROW,1..MAXROW] of record  First, Last : integer  end;
-- Cell : record  First, Last : integer  end;
-- Bucket : record
--           NumSlot, PreviousBucket, NextBucket : Integer;
--           Slots : array[1..BUCKETSIZE] of (same type as) KeyData
--         end;

PROCEDURE Initialize

OPEN "DATAFILE",1,"RANDOM"
OPEN "KEYFILE",2,"RANDOM"
NData = 0; NBucket = 0;
Zero all elements of Map

END Initialize


PROCEDURE Add

Obtain RECORD
NData += 1; KeyData.RecordPointer = NData;
WRITE 1, RECORD
Row = Hash1(KeyData.Key); Column = Hash2(KeyData.Key)
Cell = Map[Row,Column]
IF Cell.First = 0
THEN -- create a new bucket
   [Add_to_New_Bucket]
   Map[Row,Column].First = NBucket; Map[Row,Column].Last = NBucket;
ELSE -- Locate last bucket
   READ 2, Cell.Last, Bucket
   IF Bucket.NumSlot = BUCKETSIZE  -- Is the bucket full?
```

Listing 1 *(Continued on following page).*

```
   THEN -- Create a new bucket and maintain linked list
      Bucket.NextBucket = NBucket + 1; WRITE 2, Cell.Last,Bucket
      [Add_to_New_Bucket]
      Map[Row,Column].Last = NBucket -- keep track of last bucket in list
   ELSE -- Add in the same bucket
      Bucket.NumSlot += 1
      Bucket.Slots[Bucket.NumSlot] = KeyData
      WRITE 2, Cell.Last,Bucket
   END IF;
END IF;


PROCEDURE Add_to_New_Bucket

NBucket += 1; Bucket.NumSlot = 1;
Bucket.PreviousBucket = Cell.Last -- pointer to previous bucket or zero if
                                  -- this is the first bucket in the list
Bucket.NextBucket = 0 -- zero indicates end of linked list.
Bucket.Slots[1] = KeyData
WRITE 2, NBucket, Bucket

END Add_to_New_Bucket


PROCEDURE Search;
-- Seacrh through buckets.
Obtain SearchKey
Row = Hash1(SearchKey); Column = Hash2(SearchKey)
Cell = Map[Row,Column]
IF Cell.First = 0 -- sought data is definitely not on file
THEN
   DISPLAY "Data nonexistent"
ELSE
   INITIALIZE: FoundFlag = False; NextOne = Cell.First
   LOOP <BIG>
   BEGIN
      READ 2, NextOne, Bucket
      INITIALIZE: None
      LOOP <Look_in_a_Bucket>
      BEGIN for i = 1 to NumSlot
        IF Slots[i].Key = SearchKey THEN FoundFlag = True; EXIT <BIG> END IF;
      END LOOP <Look_in_a_Bucket>;
      TERMINATE: NextOne = Bucket.NextBucket
      IF  NextOne = 0 THEN EXIT <BIG> END IF;  -- when end of link is  found
                                               -- exit loop
   END LOOP <BIG>;
   TERMINATE: IF FoundFlag
              THEN
                 DISPLAY "Data in record # ";Slot[i].RecordPointer
                 READ 1, Slot[i].RecordPointer, RECORD
                 -- Perform more data processing of your choice
              ELSE
                 DISPLAY "Data nonexistent"
              END IF;

END Search
```

Listing 1 *(Continued from preceding page).*

THE RIGHT COBOL DEVELOPMENT ENVIRONMENT FOR YOUR PC

For IBM PC applications: Professional COBOL** OR

Professional COBOL anticipates many programming needs. Each function-key performs a specific thought process that automates the everyday tasks of programming IBM PC applications.

Professional COBOL gives you everything you need for total system access. Load Professional COBOL in the morning and do all your work within Professional COBOL for the rest of the day. There is no need to leave the Professional COBOL environment, even to perform DOS functions.

Professional COBOL closely couples more than 10 integrated tools and it gives you the same function-key driven command structure and help facilities in each of them.

The Professional COBOL tools include: a powerful full-screen Editor for writing code and documentation; The Micro Focus* Forms* screen painter that generates source code for interactive displays; a fast, easy-to-use Syntax Checker for full Federal High Level ANSI '74 COBOL; the unique source code Animator* which greatly simplifies debugging and maintenance; an 8086 native code COBOL Compiler to speed CPU-bound programs in your applications—you can mix intermediate code for compactness, native code for speed, and user-written assembly code for special extras; the Micro Focus multi-key B-tree ISAM for fast run-time I/O; a Run facility to execute checked and/or compiled programs; a Library facility for creating a single library file from object and reference files; and a Build facility to link your base modules into DOS loadable format.

Additional features include an on-line Help facility; file Directory facilities; the Colorizer for customizing foreground and background colors and other IBM-PC display attributes; and Run-time subroutines that let you access any DOS or BIOS function from COBOL

Professional COBOL integrates these features into a programming environment that lets you concentrate on what you really want to build. The energy and time you save can go into making software of a quality you will feel truly proud of. That's why professional COBOL benefits your productivity, and beyond that, your creativity.

*For MAINFRAME applications: VS COBOL Workbench**

Offloading mainframe program development work to the PC is now possible. The combination of Micro Focus VS COBOL Workbench with the IBM PC provides a true distributed programmer workstation.

VS COBOL Workbench allows fast and uninterrupted development, testing, and maintenance of programs downloaded to or originated on the PC. The programs are then uploaded to the mainframe for integration testing and production.

You experience no delays during program development and module testing because no one is competing for the PC's time.

VS COBOL Workbench supports most features of the COBOL language as implemented in IBM's OS/VS COBOL*, VS COBOL II*, and the Federal High Level ANSI '74 COBOL present in Micro Focus' other products.

OS/VS COBOL and VS COBOL II syntax can be used separately or coexist in a single program. VS COBOL Workbench allows easy conversion of OS/VS COBOL programs to VS COBOL II via flags that report errors in code compiled from one syntax to the other.

You can test CALLs or embedded host command languages (EXEC statements) to host database/data communications services such as IMS*, CICS*, DL/1*, SQL/DS*, and DB2*. This allows you to continue developing off-line on the PC through module and program testing.

Application testing is also made easier with the optional Session Recorder feature. The Session Recorder automatically records to disk all the keystrokes you make during a test session. You can then play back the keystrokes for regression testing. You can even edit them to keep in step with program changes.

VS COBOL Workbench closely couples an integrated set of Micro Focus tools. You switch between tools with a single keystroke. Tools such as Animator, for interactive source-level program analysis, and Forms, for fast and easy prototyping of interactive displays, greatly enhance your productivity and make your work more satisfying.

Act Now

On the West Coast, call our U.S. headquarters, 415/856-4161
On the East Coast, call our Philadelphia office, 215/668-0961

MICRO FOCUS

**Micro Focus, Professional COBOL, VS COBOL Workbench, Animator, and Forms are trademarks of Micro Focus Ltd.*

IBM, OS/VS COBOL, VS COBOL II, IMS, CICS, DL/1, SQL/DS, and DB2 are trademarks or registered trademarks of International Business Machines, Corp.

CIRCLE 84 ON READER SERVICE CARD

Recursive Procedures

By John Snyder

Over recent years, many of us in the programming business have discovered structured programming. The longer a person has been programming and the more unstructured the language environment, the more traumatic the discovery. In my case, it was an enlightenment.

I will never forget struggling through my first totally structured program (in COBOL). For a while, I thought it nearly impossible to follow the rules. But when it was done, I realized the power of the method. Regardless of the language, I have not written an unstructured program since. I have written many large programs, which executed bug free the first time they were run. I have made major modifications and additions to existing programs with relative ease and minimal damage to the original code. Obviously, I have become an advocate of structured programming.

I hope some readers can identify with my experience because this article is about a similar discovery, every bit as powerful but less well-known and less used—recursive procedures. My objective is to describe what they are and what they can do and to communicate their importance, particularly to those who have never written a recursive routine.

If you know what a recursive procedure is, but you cannot imagine ever having the need to write one—read on! You do not know what you are missing. If you do not know what a recursive procedure is, you are about to be exposed to one of the wonders of programming. Finally, I will present a small but interesting problem, which is not at all a classical recursion problem. However, after analysis, it begs for a recursive solution.

Definitions

Recursive procedures are not new, and they were not born out of programming. Recursively defined mathematical functions have been around for hundreds of years. Some of the oldest programming languages were designed with recursion in mind because it was required by the problems they addressed. For example, the LISP language was invented in 1958 by John McCarthy. Today, it remains the primary language for artificial intelligence programming. Recursion is an integral part both in data structure and function definitions.

A recursive procedure is simply a process which uses itself as a subprocess. The process has to be identified with a name, to be used for reference. In COBOL or FORTRAN, such a process would be known as a subroutine, in Pascal a procedure or function, in C a function, and in Forth a word definition. Normally, references to execute a process are known as *call*s. So, what makes a recursive procedure different is that within the code defining a process, there is a *call* to the process being defined. Sounds like infinite loop time! But, if all goes well, it isn't.

As a simple example, let us look at the factorial function, defined as follows: If n is a positive integer, n factorial, denoted $n!$, is

$$n! = n * (n-1) * (n-2) * \ldots * 3 * 2 * 1$$

(as usual, the asterisk, *, means multiplication).

In words, n factorial is the multiplicative product of all positive integers less than or equal to n.

In any programming language, it is not difficult to code a factorial process without resorting to recursion. But let us look at what recursion can do for this problem. First, we note that

$$n! = n * ((n-1)!)$$

So, if we know what $(n-1)!$ is, we can calculate $n!$ with a single multiplication. By the same token, if we knew $(n-2)!$, we could have calculated $(n-1)!$ with a single multiplication. This unraveling of the definition suggests recursion. We can create a recursive process for factorial calculation as in Listing 1.

This coding is not any particular language, just metacode. If you have never seen a recursive procedure before, study this example and play computer in your head until you develop a feel for how it works. Like structured programming, you must learn to think recursive to develop recursive algorithms.

This process illustrates several important characteristics of recursive procedures. First, note how short and sweet it is— the problem is distilled to its essence. Second, with any recursive procedure, a stopper must be provided to avoid the infinite loop. That is, there must be a condition of execution which prevents the recursive *call*, and that condition must always eventually be realized. In this case, it is *N* becoming zero, which breaks the chain of *call*s.

How they work

Programming languages either provide for recursion or they do not. Recursive procedures can be written in any programming language, but if the language does not provide for recursion, it will be a difficult task to program recursive routines.

Why is this so? Recursion depends upon two major factors. The first factor is the linkage mechanism, used when one routine *calls* another. The linkage mechanism is responsible for saving whatever is necessary in the *calling* routine, setting up access to the parameters being passed and, finally, relinquishing control to the *called* routine while storing an address to which to return when the routine has completed execution.

There are two ways to pass parameters, by *address* and *value*. Passing parameters by *address* causes the *called* routine to operate directly on the values of the parameters residing in the *calling* routine. If the *called* routine changes a parameter value, it is also changed in the *calling* routine. Passing parameters by *value* gives the *called* routine its own copies of the parameters, which may be changed independent of the parameters as they reside in the *calling* routine.

If you think about it, it is obvious that recursive routines must pass parameters by *value*. Otherwise, each successive *call* to the routine (by itself) will destroy the values in the earlier *call*s. By the same reasoning, the return address to the *calling* routine must be stored so that it is not destroyed by successive *call*s, or the *called* routine will never find its way back through the chain of linkage. Keep in mind, when a recursive routine *call*s itself, it does not load in another copy of the code to execute. The *calling* code and the *called* code are one and the same.

The second factor affecting recursion is the method of allocating storage to local variables, used only by the *called* routine. It should be clear from the discussion of parameter passing that the same space cannot be used for local variables from every *call*. Some sort of temporary storage must be allocated by the *called* routine, so that each *call* has its own copy of all local variables. In essence, recursive routines are characterized by having the same machine instructions operate on data areas which are uniquely defined for each *call*.

This should give you an appreciation for recursion's dependency on the programming language. If the linkage mechanism and local variables do not comply with the requirements of recursion, you must program your own saving and restoration of linkage, parameter, and local data items. Although this can be done by tricks within the language itself, it may be easier to write assembler-language routines to do the dirty work. For example, you could write an assembler interface between the *calling* routine and the *called* routine which allocated a temporary work area and made copies of linkage addresses and parameters.

Popular languages which do support recursion are Pascal, C, and Forth. Those which do not are BASIC, COBOL, and FORTRAN. Frankly, on a microprocessor, I believe that if you want to write a recursive routine, you should use a language which supports it. It is not worth the hassle to implement your own recursive capabilities. It might be worth it for a large application on a mainframe or minicomputer, due to other considerations in language selection.

The basic tool for allocating temporary storage is a stack. It consists of a block of computer memory and a pointer (usually a register) to indicate the area currently being used. When data is stored on the stack (commonly referred to as a *PUSH*), the pointer is automatically shifted to the next available area. You literally build a stack of data. When data is retrieved from the stack (referred to as a *POP*), the pointer is automatically shifted back so that the memory can be reused. Stacks can be used for many important programming functions, and they are the engine of recursion.

When a routine is *called* in a recursive language, the stack is used to *PUSH* all linkage data, including parameter values. The *called* routine will then allocate additional space in the stack for all local variables. For the latter function, instead of using *PUSH*es and *POP*s, the routine explicitly moves the stack pointer enough to accommodate all local variables and references the space directly. When the routine completes execution, it moves the pointer back and *POP*s the return address. Thus, no matter how many recursive *call*s are made, each *call* gets its own area in the stack.

Most microprocessors have hardware stack support. That is, they provide registers and machine instructions for performing stack operations. In Pascal and C, the use of the stack is invisible to the programmer. One routine simply *call*s another and the object code contains the stack operations.

In Forth, the data stack is explicit at the software level (linkage is handled through the *dictionary* structure, a method unique to Forth). In fact, stacks are an

```
FUNCTION FACTORIAL(N)
INTEGER N
IF  N IS NOT GREATER THAN 0  THEN
    RESULT IS  1
ELSE
    RESULT IS  N * FACTORIAL(N-1)
```

Listing 1.

integral part of the Forth language for all data storage, manipulation, and retrieval functions. If you want to learn all about what stacks can be used for, learn Forth. Stacks are one of the keys to Forth's astonishing speed as an interpretive language. Hardware stack support also simplifies assembler-language recursive programming. Lack of hardware stack support makes life difficult for assembler-language programmers and authors of compilers and interpreters.

The astute reader will have noticed a problem in the *call* by *value* discussion. Suppose the *calling* routine is passing a giant array to the *called* routine as a parameter. Is the entire array going to be duplicated in the *called* routine? In the stack?

In general, the answer is no. Arrays are passed by *address*, not by *value*, to avoid wasting storage. In C, the programmer has no choice. According to the language definition, individual variables are passed by *value*, and arrays are passed by *address*.

In Pascal, the programmer can dictate whether each parameter is to be passed by *value* (referred to, appropriately, as value parameters) or by *address* (referred to as variable parameters). But Forth is a different ball game because all *word*s are universal and parameter lists, as such, do not exist. However, everything on the Forth data stack is essentially passed by *value*, and any variables, including arrays, are passed by *address*.

Does array passing by *address* cause a problem in recursion? It possibly could, depending upon the problem, but it usually does not. If you were writing a recursive routine which required a fresh copy of an array, passed as a parameter, for each *call*, you would have to insure that each *call* had a copy with which to work.

However, it is the nature of most recursive algorithms that if an array is involved, it is something which is being scanned or manipulated at the element level from *call* to *call*, rather than being overhauled in its entirety.

Which brings us to another very useful (but not absolutely essential) recursive tool—the software pointer. The pointer allows indirect reference to a data item by an address reference rather than an explicit variable name. Pointers are efficient to use in accessing arrays because they avoid subscript calculations. They are also useful for passing arrays as parameters to *called* routines since the pointers themselves may be passed by *value*, and you avoid having to pass a subscript separately, thus saving a parameter. Pointers may be operated on arithmetically; to go to the next element of an array, you add one to the pointer for the array.

What they can do

Unlike structured programming, recursive procedures should not be used for every program. (Some would argue that not every program should be structured either, such as mainframe, on-line modules. However, that is another article.) The factorial function, for example, despite its elegance, is inefficient as a recursive procedure. The overhead of the recursive *call*s in speed and probably even memory is greater than a straightforward nonrecursive routine.

If you have encountered recursive procedures before, they were probably being used in an application with a treelike data structure, maybe artificial intelligence or system utilities, like sorting.
Trees are the classic case where use of recursive procedures is essential. Wherever you are in a tree, it looks like (and, in fact, is) the top of another tree. So you develop routines which process from the tops of trees looking down and *call* themselves when they get to the top of a new (sub)tree.

The point I'm trying to make is that recursive procedures are very useful in a wide variety of applications beyond the classic cases. Modern languages make them more accessible to the programming community. But, somewhere along the way, we forgot to teach programmers about them.

I read an otherwise excellent book on Pascal, which mentioned in only one sentence that procedures and functions could *call* themselves, but then said it would not be discussed further—that is, it was a subject beyond the scope of the book. Now, I do not expect a book about a programming language to do a dissertation on recursion, but it could at least give a hint of the significance of this capability or a reference for further study.

Recursion is simply a form of looping where each *call* is a cycle, pass, or iteration of the loop. Once you begin to think recursively, you will look at almost any program which requires looping as a candidate for a recursive procedure. I do not mean the big loops that process record-by-record from a file. I am talking about looping logic-nested loops, search loops, and particularly indefinite loops which can end at any time and/or may fail along the way and have to be redone.

The first special thing about recursive looping is that you start fresh with each iteration. You only carry the baggage from the previous iteration that you want to carry. You can concentrate on a little piece of the problem at a time, limiting the program view to the immediate data situation.

The second special thing about recursive looping is that you are actually building a chain of iterations. You do not have to complete the loop unless you want to, that is, if you are successful in what you were trying to accomplish. If you fail, you can back up in the loop as far as you want.

I am sure this all sounds very esoteric. The best way to begin to appreciate what recursion can do for you is to study some sample recursive procedures and then start developing your own. Before we launch into my example, I would like to also refer you to an ingenious recursive algorithm for sorting called Quicksort.

Unlike many recursive sorts, it does not build tree structures. It uses recursion as a looping tool, as I described earlier. You can read about Quicksort in an article called "Bubble Sort, Insertion Sort, and Quicksort Compared" by Richard G. Larson in the premier issue of *COMPUTER LANGUAGE*.

A sample problem

In this final section I will present a problem and discuss the development of a recursive solution. It is a good example of the usefulness of recursion in everyday programming. I encountered it in my work, financial application systems, not in artificial intelligence and not even in systems programming.

Suppose you have two arrays, *SUM* and *ELEMENT*, such that each entry in the *SUM* array is the sum of one or more entries in the *ELEMENT* array. In addition, when all entries in the *SUM* array are factored into sums of *ELEMENT* entries, each *ELEMENT* entry is used once and only once as a factor.

The problem is to develop an algorithm that will discover how the *SUM*s can be factored into the *ELEMENT*s. The solution may or may not be unique. That is, there may be several ways to do the factoring. However, once we have the algorithm for finding one solution, we should be able to extend it to find all possible solutions.

This is a very practical problem. Given one set of things, composed of another set of things, we wish to uncover the composition. For simplicity, I use addition as the composition method. However, very little of the algorithm will actually depend upon the composition method, and it can be translated to other well-defined methods, even nonarithmetic methods.

One of the interesting things about this problem is that it does not sound very difficult. As an exercise, once you understand the problem, you should stop reading and try to write a conventional algorithm to solve it. You will discover that it is very messy.

Obviously, the basic approach is trial and error. It may have occurred to you that sorting the *ELEMENT*s and maybe even the *SUM*s will reduce the number of trials. An *ELEMENT* entry cannot be a factor of a *SUM* entry (or what is left of a *SUM* entry after partial factoring) if the *ELEMENT* entry is greater than the *SUM* entry.

However, no matter how you try to simplify it, you can still go a long way down the road only to discover that it's the wrong road. You can factor all but the last two *SUM*s and find that what you are left with in *ELEMENT*s will not work with the remaining two *SUM*s. This means one or more of the earlier *SUM*s is not properly factored. So you need to back up and try again.

From the recursive standpoint, I look at this problem as trying to find a thread through all of the *ELEMENT*s so that when the thread is followed, it will produce all of the *SUM*s. I need to find where to start the thread, where to go next, then next, etc., factoring the first *SUM*, then continuing until all of the *SUM*s are factored. I may hit a dead end at any point and have to back up, unraveling, to try a new thread.

Each step in the attempted thread will be a recursive *call* to a routine that will find a candidate for the next factor and then continue the thread by *calling* itself again. Any *call* may fail to find a next factor, in which case I break the chain of *call*s back to the nearest point where I can try an alternate thread. If all threads are tested, eventually I will find a successful one with a chain of *call*s running through all the *ELEMENT*s. That is, when I find a solution, my depth in recursive *call*s will be equal to the number of *ELEMENT*s.

This reminds us that recursion is not necessarily cheap, especially in terms of memory. If each *call* takes *X* bytes of the stack for linkage, parameter, and local data items, and I have *Y ELEMENT*s, I need *X* times *Y* bytes of stack memory for a work area. The old trade-offs of memory vs. speed and simplicity never go away.

However, let me get back to basics on developing the algorithm. Since *SUM*s and *ELEMENT*s with a value of zero are irrelevant to this problem, I can use the convention of terminating each array with a zero value (any zeros in either array to start with must, of course, be removed). This allows me to use pointers and not have to keep track of subscripts. If I find a zero value being pointed to in either array, I know I am at the end of that array.

Next, I need a method of recording my factoring results. How do I keep track of the fact that the third *SUM* has been factored into the second, sixth, and twentieth *ELEMENT*s? I have chosen to use a *MARKER* array running in parallel with the *ELEMENT* array.

If a *MARKER* entry is zero, the corresponding *ELEMENT* entry has not yet been assigned as a factor of any *SUM*. (*MARKER* is filled with zeros to start.) If a *MARKER* entry is not zero, the corresponding *ELEMENT* entry has been assigned as a factor of the *SUM* entry subscripted by the *MARKER* value (subscripts starting with one, not zero). All *MARKER* values of one indicate factors of the first *SUM*, two, the second *SUM*, and so on. Thus, the *MARKER* array is my thread at any point in the factoring. When the factoring is completed, no *MARKER* values of zero will remain, and the *MARKER* array records the factoring.

The procedure itself, as mentioned briefly earlier, will be a routine with the job of finding a candidate to be the next *ELEMENT* factor of a given *SUM*. If it is successful, it will continue by *calling* itself again. If it fails, it will indicate its failure to the previous *call*, so it can back up and try again.

The routine will need four parameters:

■ A pointer to the current *SUM* being factored

■ A pointer to the *ELEMENT* array, positioned to the entry with which to start the search for the next possible factor of the current *SUM*

■ A pointer to the *MARKER* array, synchronized with the pointer to the *ELEMENT* array, that is, pointing to the same entry number

■ A *LEVEL* number, which is really the subscript of the current *SUM*, to use in setting entries of the *MARKER* array.

In addition, the routine needs a global reference to the beginning of the *ELEMENT* and *MARKER* arrays. As each individual *SUM* is being factored, the pointers are moved through these two arrays. However, once a *SUM* entry is completely factored, and the next *SUM* entry is to be started, the routine must have a way of returning the search pointers to the beginning of these two arrays. Note these pointers could also be passed parameters but since they are constants and would take up stack space as parameters, it is more efficient to make them globals.

Finally, the routine needs a method of returning its result. Did it find a factor or not? I have used a simple return-code switch. Zero means success, one means failure.

With this background, I can describe the routine in words:

1. Using the *ELEMENT* and *MARKER* pointers, scan for an *ELEMENT* entry which could be used as a factor of the current *SUM*. It must be unused (corresponding *MARKER* entry is zero) and less than or equal to the *SUM*. If one is not found, return failure code to previous *call*.
2. Subtract the *ELEMENT* entry factor from the current *SUM* and store the *LEVEL* number in the corresponding *MARKER* entry.
3. Increment the *ELEMENT* and *MARKER* pointers to the next entry.
4. If the current *SUM* entry is not reduced to zero, *call* the routine again using the *CURRENT* pointers and the *REDUCED SUM* value.
5. If *SUM* entry is reduced to zero, check to see if it is the last *SUM* entry. If it is the last *SUM*, return success code to previous *call*. If it is not, *call* the routine again using a pointer to the *NEXT SUM* entry, pointers to the *START* of the *ELEMENT* and *MARKER* arrays, and the *NEXT HIGHER LEVEL* number.
6. Check the result code of the recursive *call* in either step four or five. If successful, return success code to the previous *call*. If not successful, back out the processing in step two (add back the last *ELEMENT* entry to the current *SUM* and zero the last *MARKER* entry), then go to step one.

That is all there is to it! If you did try to develop a conventional routine, I am sure you can appreciate the simplicity of this algorithm. For reference, I have included the listing of this routine coded as a C language function (Listing 2). If you want to try an interesting recursion problem of your own, figure out how to extend and/or utilize this routine to find all possible factorings for a given *SUM* and *ELEMENT* array instead of just one.

Recursive procedures are fun—enjoy them! ■

John Snyder is a vice president at DISC Inc., Baltimore, Md.

```
/***********************************************************************
**                                                                    **
**              MULTIPLE ADDITION FACTORING FUNCTION                  **
**                                                                    **
**  name         factor                                               **
**                                                                    **
**  synopsis     result = factor(sum, element, marker, level);        **
**               int result;        function return code              **
**                                  0 = factoring successful          **
**                                  1 = factoring failed              **
**               int *sum;          array of sums to be factored,     **
**                                  assumed to be terminated with     **
**                                  a zero entry                      **
```

Listing 2 *(Continued on following page).*

PROFESSIONAL PROGRAMMER'S BULLETIN:

Be Productive, Be

BRIEF™

TRY BRIEF "RISK-FREE" FOR 30 DAYS WITH OUR MONEY-BACK GUARANTEE!

The Programmer's Editor

BRIEF's power and flexibility provide dramatic increases in programming productivity. BRIEF's ergonomically designed human interface becomes a natural extension of your mind, allowing you to eliminate tedium and concentrate on creativity.

- WINDOWS
- Full UNDO (N Times)
- Compile within BRIEF
- Keystroke Macros
- Exit to DOS inside BRIEF
- Programmable Macro Language
- Multiple files, unlimited size
- "Regular Expression" search
- Reconfigure keyboard
- Language sensitive user controllable features (such as Auto-Indent for C)

AVAILABLE FOR PC-DOS, IBM-AT, AND COMPATIBLE SYSTEMS

ONLY $195.

DEMO AVAILABLE FOR ONLY $10 (applicable to future purchase)

CALL TOLL FREE 800-821-2492 for "Technical Description" or to order.

Solution Systems™

335-L Washington St., Norwell, MA 02061
617-659-1571

BRIEF is a trademark of UnderWare.
Solution Systems is a trademark of Solution Systems.

CIRCLE 37 ON READER SERVICE CARD

Pascal and C Programmers

Your programs can now compile the *FirsTime*™

FirsTime is an intelligent editor that knows the rules of the language being programmed. It checks your statements as you enter them, and if it spots a mistake, it identifies it. *FirsTime* then positions the cursor over the error so you can correct it easily. *FirsTime will identify all syntax errors, undefined variables, and even statements with mismatched variable types.* In fact, any program developed with the *FirsTime* editor will compile on the first try.

Unprecedented
FirsTime has many unique features found in no other editor. These powerful capabilities include a zoom command that allows you to examine the structure of your program, automatic program formatting, and block transforms.

If you wish, you can work even faster by automatically generating program structures with a single key-stroke. This feature is especially useful to those learning a new language, or to those who often switch between different languages.

Other Features: Full screen editing, horizontal scrolling, function key menus, help screens, inserts, deletes, appends, searches, and global replacing.

Programmers enjoy using *FirsTime.* It allows them to concentrate on program logic without having to worry about coding details. Debugging is reduced dramatically, and deadlines are more easily met.

FirsTime for PASCAL	$245
FirsTime for C	$295
Microsoft PASCAL Compiler	$245
Microsoft C Compiler	$395
Demonstration disk	$25

Get an extra **$100 off** the compiler when it is purchased with ***FirsTime.***
(N.J. residents please add 6% sales tax.)

Spruce
Technology Corporation
110 Whispering Pines Drive
Lincroft, N.J. 07738
(201) 741-8188 or (201) 663-0063

Dealer enquiries welcome. Custom versions for computer manufacturers and language developers are available.

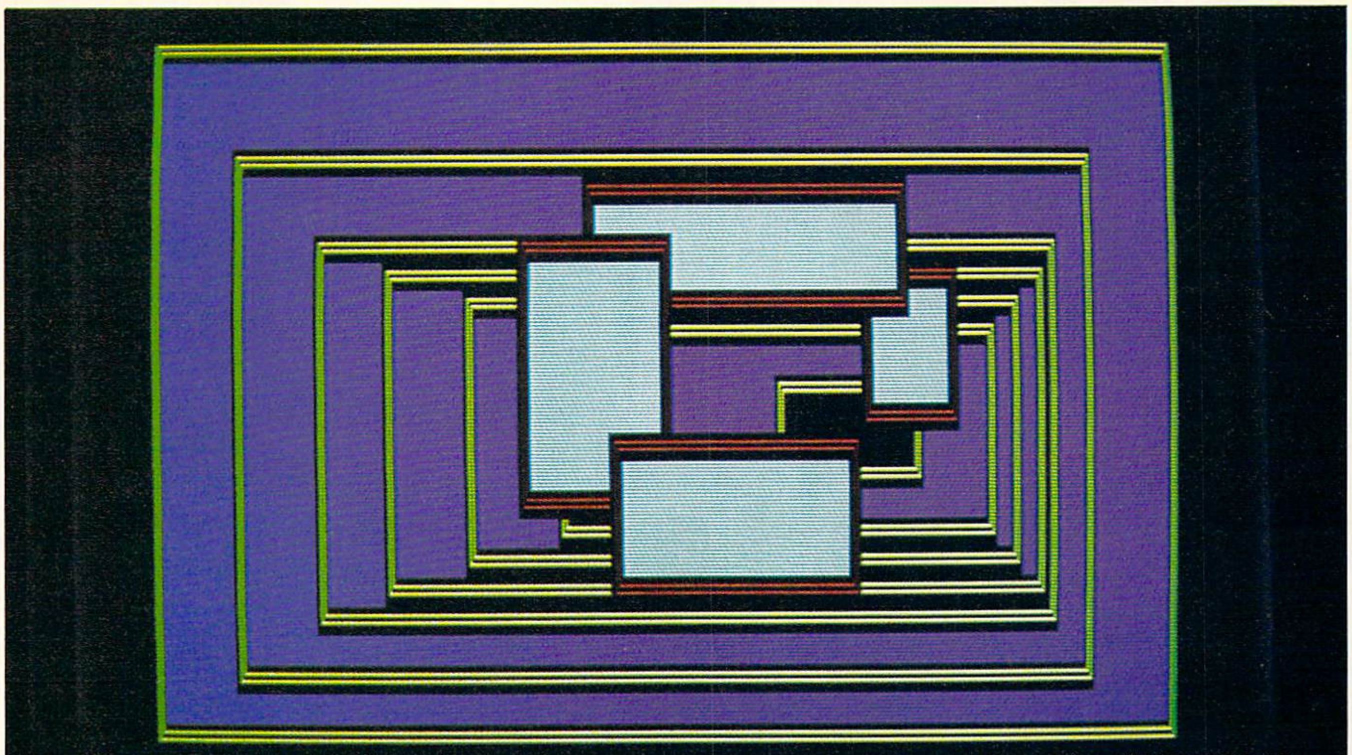
FirstTime is a trademark of Spruce Technology Corporation.

MasterCard VISA

CIRCLE 33 ON READER SERVICE CARD

```
**              int *element;    array of factors elements,      **
**                               assumed to be sorted in         **
**                               ascending numerical sequence    **
**                               and terminated with a zero      **
**                               entry                           **
**              int *marker;     array of factor markers         **
**              int level;       current sum (subscript) number  **
**                                                               **
**   globals    int *starte;     constant pointer to first       **
**                               entry of element array          **
**              int *startm;     constant pointer to first       **
**                               entry of marker array           **
**                                                               **
**   description Factor function will find the elements which    **
**              comprise the sums by recursive "threading"       **
**              analysis.  Each call will try to identify an     **
**              entry in the element array which can be used     **
**              as an additive factor of the current entry in    **
**              the sum array.  If the search is successful,     **
**              the factoring is continued by a recursive        **
**              call, until all sum entries are factored.        **
**              Each entry in the element array must be used     **
**              once, and only once, in the total factoring      **
**              of all sums.                                     **
**                                                               **
*****************************************************************/
/** GLOBALS TO START OF ELEMENT AND MARKER ARRAYS              **/
int *starte, *startm;
/** START OF FUNCTION                                          **/
factor(sum, element, marker, level)
int *sum, *element, *marker, level;
{
    /** LOCAL DATA DECLARATION                                 **/
    int result;
    /** MAIN SEARCH LOOP                                       **/
    for ( ; ; ) {
        /** FIND AN UNUSED, POSSIBLE FACTOR OF SUM             **/
        for ( ; *marker && *element <= *sum; element++, marker++)
            ;
        /** CHECK FOR ELEMENT TOO BIG OR END - FAILURE         **/
        if (*element > *sum || *element == 0)
            return(1);
        /** FOUND ONE - DEDUCT FROM SUM AND MARK IT            **/
        *sum -= *element++;
        *marker++ = level;
        /** IS SUM COMPLETELY FACTORED ?                       **/
        if (*sum)
            /** IF NOT - GO FOR NEXT FACTOR                    **/
            result = factor(sum, element, marker, level);
        else
            /** IF SO - IS IT THE LAST SUM ?                   **/
            if (*(sum+1) == 0)
                return(0);
            else
                /** IF NOT - START ON NEXT SUM                 **/
                result = factor(sum+1, starte, startm, level+1);
        /** WAS THREAD SUCCESSFUL ?                            **/
        if (result == 0)
            return (0);
        /** IF NOT - BACK OUT LAST FACTOR AND CONTINUE         **/
        *sum += *(element-1);
        *(marker-1) = 0;
    }
}
```

Listing 2 *(Continued from preceding page).*



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?)

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:

$59.95

- *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

CI.

AMBER
AMBER SYSTEMS
1171 S. Saratoga-Sunnyvale Road
San Jose, CA 95129

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

Sorting by Dispersion

By David Keil

Computer journals have recently published a number of articles on sorting algorithms. The authors have usually favored the comparison method used in the Shell sort and in C.A.R. Hoare's Quicksort. Some have noted as well the existence of the distribution or dispersion method, which also goes under the names of hash coding, address calculation, bulldozer sort, and range sorting.

Sorts using the comparison method arrange elements of an array, such as words, in ascending lexicographical order by repeated comparisons of selected elements, two at a time. The processing exchanges elements that are found to be out of order with respect to each other. After the program has compared each element with a certain number of other elements in the array and made the appropriate exchanges, it can be determined that the entire array has been sorted.

An alternative approach

The dispersion method makes an initial pass through the array, evaluating each element in some way and putting a vector into internal storage at a location corresponding to the value of the element. This vector points to the element.

The algorithm then retrieves all the pointers in their new, sorted order. That is the procedure, roughly sketched.

We may try to sort a small mailing list according to zip code, for example, by assigning each record to one of a thousand pointers. The subscript in the pointer array will be the value of the first three digits of the zip code in the corresponding record.

After the pointer array has been created, we may run through it, collecting nonzero pointers and thus gaining access to the records they point to. They will be in perfect order by the first three digits of the zip code.

If two records' zips have the same first three digits, however, we have a problem: room must be made for two pointers that, according to the rules, should have the same subscript. One solution is to assign the record to a location one higher or one lower than the corresponding three-digit zip value. Additional displacements may be made in case there is still a collision.

Provided the number of such collisions is low, the extra time required to move a pointer up or down the array—before depositing it where it will lie until collection time—will be well spent, because a dispersion-based sort can be very fast, as will be seen later on.

But if our mailing list happens to be entirely from the same region, the large number of collisions will cause the sort to take a very long time—longer than the Shell sort or Quicksort.

A possible solution to the collision problem is to create a very large array with plenty of room for colliding pointers. Instead of shifting hundreds of pointers up or down the pointer array to make room for one new pointer, we might displace a pointer into one of the extra slots, of which there will be plenty.

If we were manually sorting index cards with names on them, we might use such a method. We could put cards on a table with plenty of space between them to make room for yet-to-be-encountered cards.

Such a solution entails plenty of table space—or, in a computer, high-memory overhead. Harold Lorin's *Sorting and Sort Systems* (p. 152) states that for certain implementations, "the algorithm requires space for the representation of $2(A*N)$ elements," where N is the size of the array being sorted and A is the number of possible values for a character position. (If numbers were being sorted, A would be 10; if words composed of letters of our alphabet were being sorted, A would be 26.)

In other words, if we were sorting 50 index cards into groups according to the first letter of the name on each card, we would need table space for 26 times 50, or 1,300 cards, to cover the possibility that all the cards will be in the same group.

Sometimes data that must be sorted has more or less randomly-distributed key values. For these, the simple "address-calculation" sort described by Douglas Davidson in *BYTE* (November 1983) will be adequate.

But if we are sorting names, text words, or even zip codes, we will almost certainly find a highly nonrandom distribution. We will find clumps of Smiths and Jones's that will cause large numbers of collisions and hence turn a fast sort into a slow one.

This article will present a way to solve the problem of quickly sorting unevenly distributed data in a memory space not substantially greater than that required to hold the array being sorted. In other words, it will describe a way to avoid the time-consuming processing of collisions without using a large extra amount of memory.

Why dispersion can sort faster

First let us examine the comparison approach and see what its limits are. The graph in Figure 1 indicates how the Shell sort, a fast, comparison-based algorithm, requires increasing amounts of time per element to sort increasingly large arrays.

Because the *X* coordinate (Size of array (*N*)) on the graph is scaled logarithmically, with a base of two, we can easily see that the Shell sort requires an execution time proportional to $\log_2 N$ to process each element; the curve approximates a straight line originating at zero.

As Donald Knuth has shown, sorting algorithms based on comparisons cannot execute faster than this—the total time is at best proportional to $N*\log_2 N$. (That is, at least $\log_2 N$ comparisons must be made for each element.)

This is because a comparison yields only one bit of information: either the left element is greater than the right or it is not; either an exchange must be made or no exchange is to be made. For each doubling of the size of the array to be sorted, an extra pass is required, comparing each element to some other element.

By the time we're sorting about 16,000 elements, about 13 passes through the whole array will be necessary: $2^{13} = 16{,}384$. The way to flatten the graph and decrease the number of passes necessary is to extract more than one bit of information from each examination of an element.

String data yields eight bits of information just by an examination of the first eight-bit character. Words composed of letters of our alphabet yield about five bits by such an examination; that is, the first letter can be anything from A to Z if capital and lower-case letters are assigned the same value. 26, the number of letters in our alphabet, is approximately 2^5.

Increasing in this way the number of bits of information extracted in examining an element means that instead of needing an extra pass each time the array size is doubled, we will require an additional two-way pass only after the size of the array increases 26-fold.

In other words, if we extract information from the first character of a word string, rather than comparing strings, our goal can be a sort-time-per-element figure proportional to $\log_{26} N$. For 17,000 elements, $\log_{26} N$ is only about 3. This means that six passes are necessary for such a sort: three to distribute elements and three to collect. Six passes are fewer than half as many as 13, the number of passes required to sort 16,000 elements with a comparison-based algorithm.

Test results

Observe Figures 1 and 2. Here we find that the performance of some dispersion-based algorithms (called P.DISP in Figure 1 and DISP.1 and DISP.2 in Figure 2) have slopes that are consistent with this general theoretical prediction. The dispersion algorithm sorts large numbers of random array elements almost twice as fast as the Shell algorithm.

If we extrapolated the performance figures shown, we could predict that tens or hundreds of thousands of random elements can be sorted, using dispersion, in only slightly more time per element than is necessary to sort hundreds of elements. That is, total sorting time is very close to being proportional to the number of elements sorted.

The dispersion sort illustrated not only performs roughly twice as fast as Shell in the 3,000-to-6,000-element range, it promises to perform better and better in comparison with Shell or any other comparison-based sort as the number of elements to be sorted rises.

It is moreover not adversely affected by preordering of arrays (as is Quicksort), and it is not highly sensitive to unevenly distributed data. Compare DISP.2 sorting text data in Figure 2 with the two sorts of random data. (DISP.1 and DISP.2 are similar dispersion sorts.) The strings used in the text data were the words from a scene in "Hamlet, Prince of Denmark" whereas the other samples were randomly generated, three-letter strings. Performance with nonrandom word strings was not qualitatively slower than with random strings in the 50-to-1,600-element range.

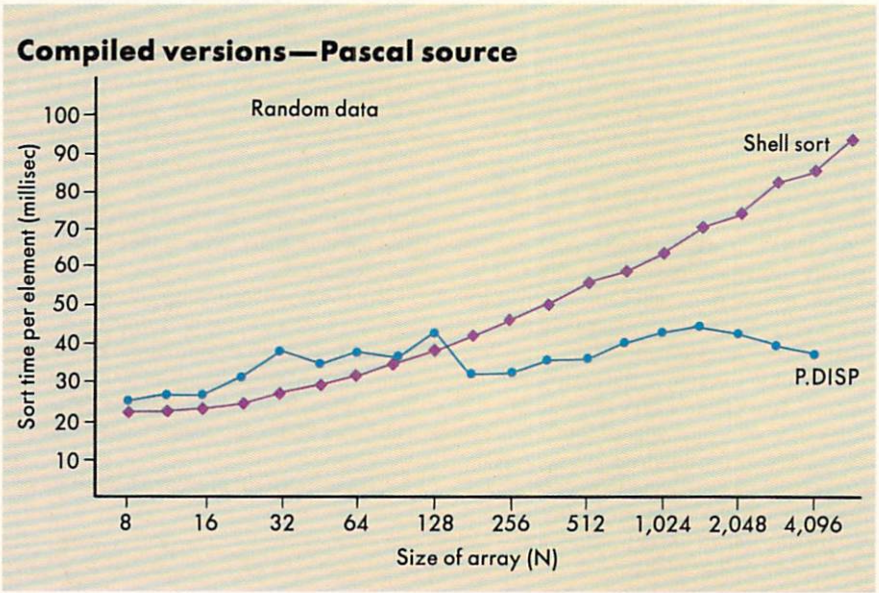


Figure 1.

This dispersion algorithm is thus a general purpose one, requiring no special inspection of the data before use. It is designed in such a way that its worst-case performance will be comparable to the performance of the Shell sort, and its best-case performance will be much, much faster than Shell.

One dispersion algorithm

The Pascal listing, P.DISP, presented in Listing 1 shows the logic of one of the algorithms whose performance was graphed in Figure 2. Exactly the same logic was used in the BASIC program DISP.1, and the logic in DISP.2 was almost identical.

Five modules of P.DISP are relevant to our discussion: the main routine; the segment-processing routine, *ProcSeg*; the pointer-dispersing routine, *Distribute*; the pointer-collecting routine, *Collect*; and the Shell-sort routine, *Shell*, which is used as part of the dispersion algorithm. Two other procedures, *Create* and *Report*, come into use only to generate random string data, time the test, and display the results.

To sort our array of random three-letter words, the program first examines each element and puts it into one of 27 "bins" where it can be retrieved. P.DISP does this by beginning with a segment consisting of the entire array. *ProcSeg* processes this segment, first calling *Distribute* to allocate the elements among 27 linked lists according to the letter of the alphabet with which the word begins. An extra twenty-seventh bin will accept words not beginning with a letter of the alphabet.

The linked lists are one key to the success of the algorithm. Instead of creating 27 arrays, each dimensioned to the maximum number of elements to be sorted, *N*, our algorithm creates one array of 27 elements, called *Anchor*, and an array of *N* integer links. *Anchor(1)* is a pointer pointing to the first element beginning with "a"; *Anchor(2)* points to the first "b" word, etc.

To insert a second "a" word into the linked list, the word's subscript is assigned to *Anchor(1)*. *Anchor(1)* acquires a link that points to the previous occupant of *Anchor(1)*. Further additions to the "a" bin are accomplished by replacing *Anchor(1)* and by linking the new occupant of *Anchor(1)* to its old occupant. The "a" bin, a linked list, has the capacity to contain all the elements in the array to be sorted, if necessary. Twenty-seven such linked lists are created by *Distribute.*

A way to understand the linked lists in P.DISP is to picture a row of 27 hooks on the wall. We hang an "a" word from the "a" hook. The "a" word has a hook hanging from it as well. To add a new "a" word, we take the top word off the hook on the wall, replace it with the new word, and hang the old top word from the new word's hook. More words are added in the same way, at the top.

Why add words at the top and not at the bottom of the chain? Because to access the bottom of a linked list in computer memory, we must examine each link to learn what the next element will be. Adding to the top allows us to avoid a possibly long series of examinations each time a new element is added to the list.

This brings us to the next step of the dispersion sort, the collection routine—*Collect* in the Pascal program illustrated here. Again, each chain is removed from the wall and the words are taken off the top of the chain in order.

The linked lists here are employed as stacks, not as queues; the last "a" word added is the first retrieved. In this way the collect module may retrieve each element in one short step. As the bins are emptied, A through Z, their pointer contents are assigned to an *N*-element pointer array.

If each bin has only one element in it, or

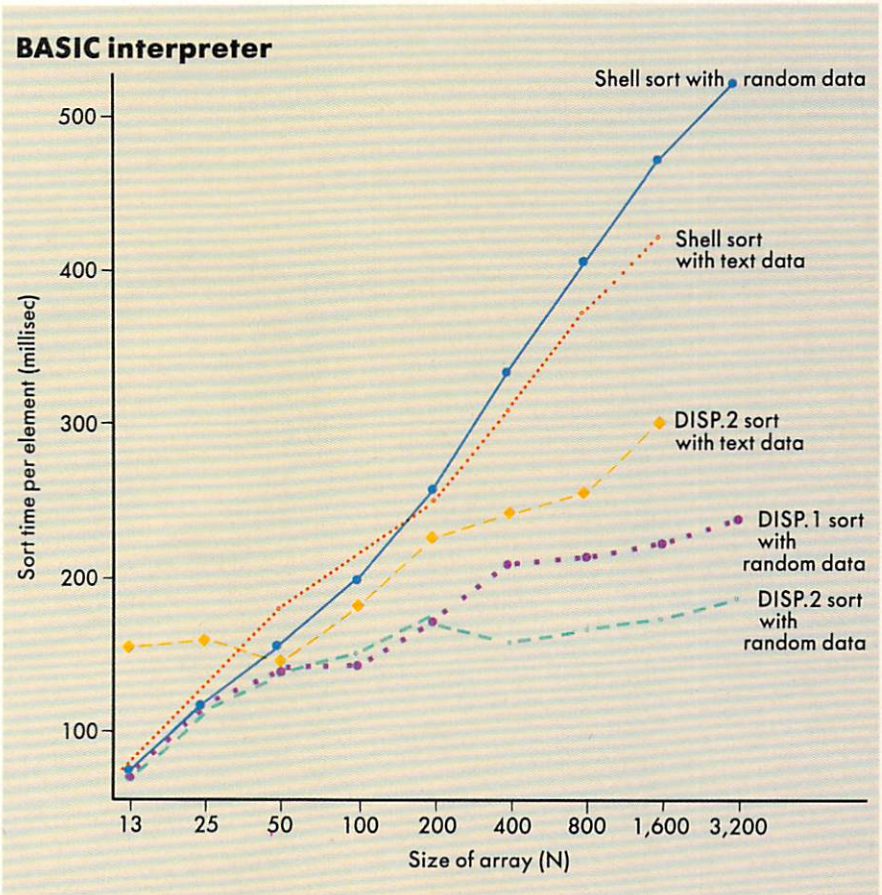


Figure 2.

none, our entire array has been sorted in one step. If not, we must perform further processing, because *Distribute* and *Collect* do not sort within bins, only among them.

To sort within a bin, as is required whenever the bin contents number more than one, our algorithm selects one of two paths. If a bin's size is less than nine, *Collect* calls *Shell*, which sorts the contents of the bin by the comparison method. Nine is an arbitrary choice. We use Shell here because it sorts small arrays faster than a dispersion sort. (A Bubble sort in place of the Shell sort might sort the nine or fewer element bins in even less time.)

If the size of the bin is greater than nine, we consider it, after retrieval, as a segment, saving its beginning and end locations in the overall pointer array as LO and HI for that segment.

The depth at which the program is examining the data array— one if the first letter of each element is being evaluated, two if the second, etc.—is saved as DEPTH for the new segment. Then we continue collecting the contents of bins until the twenty-seventh is emptied.

On completion of *Collect*, control proceeds to the next segment. If the first segment was the entire array to be sorted, the second is the first bin that contains more than nine elements. Just as we have sorted the entire array into 27 bins in processing the first segment, the processing of the second segment sorts one of the resulting bins into 27 bins.

Our program distributes and collects the contents of this second segment, examining the second character of each word. If the "b" bin was the first to contain more than nine elements on the first pass, then "bat," "baffle," and "bar" will end up in the first bin, the "a" bin, of this second distribution and collection.

Through use of the method known as tail recursion, the entire array is repeatedly subdivided in this way into bins until it has been fully sorted.

If a bin is collected that results from a distribution on the third character and still has more than nine pointers in it, then it is Shell sorted rather than saved as a segment to be distributed and collected on the fourth character. There is no point in trying to distribute and collect on the fourth character because our random test data words contain only three characters each.

If all the elements in a segment fall into the same bin, then we Shell sort that bin rather than distribute it again, because we are in a situation where it seems likely that examinations of the next character bin will only yield more bins containing all the elements, as would be the case with a long list of Smiths.

Using a range table

We have examined here only one way to use the dispersion technique in a general purpose sort. Improvements are not hard to imagine; one example is a range table that allocates words among bins according to an examination of two or three characters at a time.

Calculating the bin number for a distribution on two characters would require 27*27, or 729 bins, and most would probably end up empty, such as all the bins containing words beginning with "x" and another consonant. A much smaller number of bins would be needed if a good range table were used, because only a small proportion of these 729 combinations of letters actually occur in English-language names and text words.

Rather than yielding four or five bits of information, as an examination of one letter does, an examination of two letters, using a range table to choose a bin number, would yield perhaps eight bits (256 bins). In that case, we might aim for a much greater speed than is possible with the algorithm presented in P.DISP. Whereas P.DISP requires six passes to sort 16,000 randomly chosen strings, up to 64K English-language words could be sorted in four passes if the range table yielded eight bits of information per examination.

The dispersion method is known to the computer world but seems to have been rarely applied with microcomputers due to the greater complexity of the algorithms, the higher memory overhead (a factor sometimes overestimated), and the only marginal speed advantage at array sizes below about a thousand. An additional factor may be that fast algorithms are undoubtedly often kept as business secrets.

As hard disks become more widespread and RAM chips of greater capacity come into use, it seems likely that dispersion-based sorting algorithms will tend to replace algorithms based on comparison. In spelling-checker programs, for example, a fast sort of words in RAM will mean a very fast check and much-improved response, to the benefit of users. That is part of what the present generation of computers and software is all about. ■

David Keil works in computer interfaced typesetting at Crockergraphics, Needham, Mass. He has a B.A. in history from the Univ. of Minnesota.

```
PROGRAM P.DISP  (DISPERSION SORT);
(**)
VAR L,E,B,I,X,POINTER,SEGMENT,LAST,T1,T2,T3,LETTER: INTEGER;
    TI: REAL;
    P,LINK: ARRAY[1..4096] OF INTEGER;
    ANCHOR: ARRAY[1..27] OF INTEGER;
    HI,LO,DEPTH: ARRAY[1..60] OF INTEGER;
    ELEMENT: ARRAY[1..4096] OF STRING[3];
(* CREATETESTDATA *)
PROCEDURE CREATE (X: INTEGER);
  BEGIN
  MEM[$D021]:= CHR(0);
  WRITE(CHR(147),CHR(5),CHR(14));
  FOR I := 1 TO 27 DO
    ANCHOR[I] := 0;
  WRITE('NO. OF ELEMENTS? ');
  READ(L);
  FOR I:= 1 TO L DO
    BEGIN
```

Listing 1 *(Continued on following page).*

```
    T1:= TRUNC(65+26*ABS(RND(1)));
    T2:= TRUNC(65+26*ABS(RND(1)));
    T3:= TRUNC(65+26*ABS(RND(1)));
    ELEMENT[I]:= CONCAT(CHR(T1),CHR(T2),CHR(T3));
    WRITE(ELEMENT[I],' ');
    P[I]:= I;
    LINK[I] := 0;
    END;
  MEM[$00A0] := CHR(0); MEM[$00A1] := CHR(0); MEM[$00A2] := CHR(0);
  SEGMENT:= 0; LAST:= 1; LO[1]:=1; HI[1]:= L; DEPTH[1]:= 1;
  END;
(* DISTRIBUTE ELEMENTS *)
PROCEDURE DISTRIBUTE (B,E: INTEGER);
  VAR WORD: STRING;
      SEGDEPTH: INTEGER;
      LETCHAR: CHAR;
  BEGIN
  SEGDEPTH := DEPTH[SEGMENT];
  FOR I:= B TO E DO
    BEGIN
    LETCHAR := COPY(ELEMENT[P[I]],SEGDEPTH,1);
    LETTER := ORD(LETCHAR)-64;
    LINK[P[I]] := ANCHOR[LETTER];
    ANCHOR[LETTER] := P[I];
    END
  END;
(* SHELLSORT *)
PROCEDURE SHELL (B,E: INTEGER);
  VAR D,SL,J,T,FIRST,SECOND: INTEGER;
  BEGIN
    SL:= E-B+1;
    D:= TRUNC(EXP(TRUNC(LN(SL)/LN(2))*LN(2)))-1;
    WHILE D>=1 DO
      BEGIN
      I := B;
      WHILE I<=E-D DO
        BEGIN
        J := I;
        WHILE J>=B DO
          BEGIN
          FIRST := P[J]; SECOND := P[J+D];
          IF ELEMENT[FIRST] > ELEMENT[SECOND]
            THEN BEGIN
                 T:= P[J]; P[J]:= P[J+D]; P[J+D]:= T; J:= J-D;
                 END
            ELSE J:= 0;
          END;
        I:= I+1
        END;
      D:= TRUNC(D/2);
      END;
  END;
(* COLLECTELEMENTS *)
PROCEDURE COLLECT (BEGSEG: INTEGER);
  BEGIN
  I := BEGSEG-1;
  FOR LETTER:= 1 TO 27 DO
```

Listing 1 *(Continued on following page).*

```
    BEGIN
    B:= I+1;
    IF ANCHOR[LETTER] > 0 THEN
      BEGIN
      I := I+1;
      P[I] := ANCHOR[LETTER];
      ANCHOR[LETTER] := 0;
      WHILE LINK[P[I]]> 0 DO
        BEGIN
        I := I+1;
        P[I] := LINK[P[I-1]];
        LINK[P[I-1]] := 0;
        END;
      IF I-B > 9
        THEN
          IF (B = BEGSEG) OR (DEPTH[SEGMENT] > 2)
            THEN
              SHELL(B,I)
            ELSE
              BEGIN
              LAST:= LAST + 1;
              LO[LAST]:= B; HI[LAST]:= I; DEPTH[LAST]:= DEPTH[SEGMENT] + 1;
              END
        ELSE
          IF I > B
            THEN SHELL(B,I);
      END
    END
  END;
(* PROCESSSEGMENT *)
PROCEDURE PROCSEG (X: INTEGER);
  BEGIN
  WRITE(SEGMENT,' ');
  DISTRIBUTE (LO[SEGMENT],HI[SEGMENT]);
  COLLECT (LO[SEGMENT]);
  END;
(* PRINTRESULTS *)
PROCEDURE REPORT (L: INTEGER);
  BEGIN
  T1 := ORD(MEM[$00A0]); T2 := ORD(MEM[$00A1]); T3 := ORD(MEM[$00A2]);
  TI := 1092.26*T1+4.26667*T2+T3/60;
  WRITELN(CHR(13),L,' ELEMENTS',CHR(13),'TIME: ',TRUNC(100*TI+0.5)/100);
  WRITELN('MILLISEC/ELE: ',TRUNC(1000*TI/L+0.5));
  WRITELN('PRESS RETURN'); READ(X);
  FOR I:= 1 TO L DO
       WRITE (ELEMENT[P[I]],' ');
  END;
(* MAIN ROUTINE *)
BEGIN
  CREATE(0);
  WHILE SEGMENT < LAST DO
    BEGIN
    SEGMENT:= SEGMENT + 1;
    PROCSEG (0);
    END;
  REPORT(L);
END.
```

Listing 1 *(Continued from preceding page).*

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.

CIRCLE 53 ON READER SERVICE CARD

mbp COBOL. $1000

Please send complete mbp COBOL information to:

NAME ______________________

COMPANY ______________________

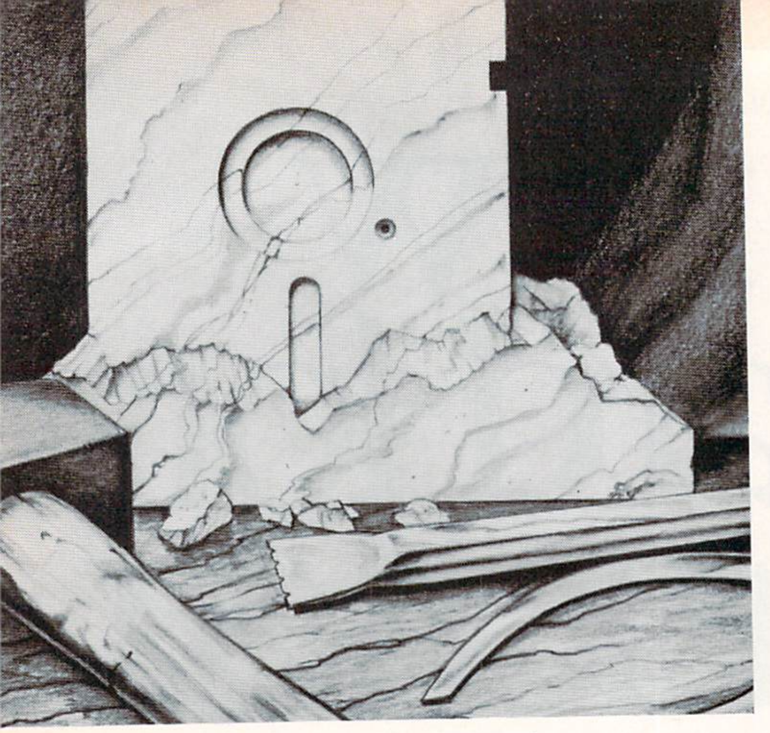
ADDRESS ______________________

CITY/STATE/ZIP ______________________

PHONE ______________________

mbp Software & Systems Technology, Inc.
7700 Edgewater Drive, Suite 360
Oakland, CA 94621
Phone 415/632-1555

mbp®



"In the art of programming the difference between greatness and mediocrity is often the quality of the artist's tools."

POLYOVERLAY

Make Maximum Use of Memory with Overlay Code. Creates optimal memory overlay structures, in terms of utilizing a minimum of total physical memory, while it builds a batch file to automatically drive an overlay linker, for creating efficiently overlayed executable code. Creates optimum overlay structures for any modular language including C, FORTRAN, Pascal, Ada and even BASIC. Essential for porting mainframe code too large to fit into personal computer's memory all at one time. Requires MS-DOS 2.0, and Microsoft's Linker Version 2.4 Shipped with Users Manual **$99**

POLYTRON C LIBRARY 1

A Library of High-Performance Functions for Lattice C Compiler Users. Over 50 significant routines more useful than common library functions. The library includes UNIX-like utility functions, file management functions, more efficient and flexible I/O routines, and executive functions. This is a powerful, professional-quality C library that is easy to use and designed to increase your productivity. Assembly and C source code included. Requires Lattice C Compiler Includes Users Manual **$99**

POLYLIBRARIAN

The Object Module Library Manager for MS-DOS™. PolyLibrarian lets you create a single library from related Objects Modules. The IBM PC Linker will then select only the modules necessary to produce an executable file. PolyLibrarian can: Create, organize, reconstruct, dissect, and compress libraries of object code modules. Add, delete and replace an object module. Extract a module from a library to create an OBJ file. Change public and external names. Modules are time-stamped within the library. Supports full path names. Supports Microsoft's original and latest Linker (Version 2.4). Includes installation program to customize options to your specific needs. Features three Modes of Operation (Interactive, Command File, Command Line) that can be freely intermixed. Surpasses performance of other primitive librarians. Features a highly functional user interface that uses simple meaningful key words. Requires 128K RAM, MS-DOS (PC-DOS) 1.1 or 2.0. Compatible with any compiler or assembler that uses the MS-DOS (PC-DOS) Linker. Shipped with comprehensive Users Manual including step-by-step tutorial for **$99**

POLYLIBRARIAN II

Extends PolyLibrarian's power to Intel™ and Digital Research™ Format Libraries. PolyLibrarian II is an enhanced version of PolyLibrarian. In addition to all the features and functions listed above for PolyLibrarian, this version adds a new dimension of productivity for programmers using Intel and Digital Research Language Products. Compatible with any Intel Format Library and any Microsoft Format Library. Compatible with Intel's Link 86 under MS-DOS. Compatible with Digital Research Linker under MS-DOS **$149**

POLYMAKE

The Intelligent Program Builder & Maintenance Tool for MS-DOS. PolyMake frees you from the need to remember which files depend on others and which files have been modified. PolyMake will: Remember the exact sequence of operations necessary to make a new version of a program. This will significantly reduce developement time, prevent bugs & manage large software projects. Automatically invoke your compiler, assembler, linker, librarian, or do whatever is necessary to bring all dependent files up to date. Compare the date and time of all relevant files and use internal rules to rebuild a program or complete software system. Rules can be modified and expanded. Extensive debug capabilities help construct efficient make files. Includes features superior to UNIX Make. Fully automatic operation under MS-DOS 2.0. Full path name capabilities under MS-DOS 2.0. Requires 128K. Shipped with a comprehensive Users Manual **$99**

POLYFORTRAN TOOLS 1

Four Powerful Utilities in One Package. For developing or enhancing software or porting mainframe software to personal computers these easy-to-use tools will become indepensible for the professional FORTRAN programmer. Includes: 1. **FORTRAN-XREF** for developing comprehensive cross-reference maps of FORTRAN source code. Mainframe users expect this kind of utility. Now personal computer users have X-REF power. 2. **FORTRAN-EXTRACT** extracts the MAIN routine as well as all SUBROUTINES, FUNCTIONS and BLOCKDATAS from FORTRAN source code. It is unsurpassed for breaking-up large codes into manageable modules that can be individually modified, compiled, added to libraries and finally linked into a single executable code. 3.**FORTRAN-CLEANUP** performs three critical functions to improve FORTRAN source code readability: A. Renumbers FORTRAN souree code statements within each routine to begin at a user specified starting number; B. Adds spaces around delimiters and key words; C. Performs indentation for DO loops, IF...Then and similar structures. If you are becoming weary of cleaning up "sloppy" code then you need FORTRAN-CLEANUP. 4. **FORTRAN-SCREEN** lets you show off your software. This is a complete set of FORTRAN callable screen attribute and cursor position control functions for creating user friendly, interactive menus and displays. Requires MS-DOS 2.0, 256K, One Disk Drive, Compatible w/hard disk. Shipped with Users Manual **$179**

Praise From Professionals

"PolyLibrarian is a powerful tool for serious hobbyists and professional programmers. It is a thing of beauty; a work of art. The documentation is excellent, the large selection of commands are intuitive to learn and easy to use, and the program itself provides a service of inestimable value. It is refreshing to see a tool of this caliber available for serious programmers. The design of this program shows foresight and ingenuity. It sets the human engineering standard for programming utilities. If all of a programmer's tools were so simple to work with, better programs could be developed in less time and with fewer headaches. *Bravo, Polytron!* I look forward to your next product."

Dan Rollins,
Programmer and Author writing in PC Age

"PolyLibrarian is an extensive, friendly utility . . . an excellent tool for serious programmers. It combines professional quality with a flexible user interface."

Greg Estes
Editor, Programmers Journal

"I am thoroughly pleased [PolyLibrarian] is definitely one of the best products of its type that I have ever used on any system."

An Unsolicited Comment from
Steve Kauffman
Consulting Engineer

To Order or Request Literature Call

1-800-547-4000

Ask For Dept. 315
VISA & MasterCard Accepted
Or Send Checks, P.O.'s To:
Polytron Corporation DS-315-F1
P.O. Box 787, Hillsboro, OR 97123
Add $1.25 Shipping Charge For Each Product Ordered.

CIRCLE 88 ON READER SERVICE CARD

TM POLYTRON

Part III: Translating tables

Programming Macros in C

By Alexander B. Abacus

>>>>>>>>
>>>>>>>>
>>>>>>>>
>>>>>>>>

In this final installment of my three-part series on C macros, we'll explore how the preprocessor of a C compiler can be used to translate a common representation of a table of data into different but related tables required by different programs. The technique is applied to the generation of structure definitions and corresponding initialization lists in the C programming language.

Programming problems often call for table-driven solutions. Related programs or different modules of the same program may use tables derived from the same data. If separate copies of the tables are built into different parts of programs, multiple copies of tables must be synchronously maintained. A better way is to keep one master copy of data and generate all related tables from it.

For programs written in C and for certain classes of tables, the preprocessor can be used to generate variations of tables. Here we explore one way to do it. We first concentrate on a specific problem. Later we generalize the technique.

Structure initialization

When initializing a structure in C, the list of initial values follows after all names of structure members have been specified, as in Listing 1.

In larger structures it is difficult to establish correspondence between the member name and the initial value. It would be more convenient if we could write:

```
struct tag
{
     char * member1 = "first";
     char * member2 = "second";
} instance;
```

Let us try to use the C preprocessor to generate the first form from specifications similar to the second form. We chose the following format to specify the initialized structure:

```
StructBEGIN(tag)
 DCL(char * member1) INIT("first")
 DCL(char * member2) INIT("second")
StructEND(instance)
```

where *StructBEGIN()*, *StructEND()*, *DCL()*, and *INIT()* are preprocessor macros to be defined.

We can easily define the macros to generate from this specification either the first part of the required text (declaration) or the second part (initialization), shown in Listing 2.

Implementation constraints

Our goal is to generate both declaration and initialization parts from one copy of the compact specification. To achieve this goal using only the C preprocessor, we have to impose certain severe constraints on our idealized solution.

Each specification of initialized structure must be kept in a separate *include* file, and each file must be included twice on two consecutive lines in order to generate the complete initialized structure. Therefore, this approach becomes practical only for large structures. An *include* file defining an initialized structure must have the form shown as follows:

```
/* File aStruct.h—begin. */

#include "initstruct.h"
```

```
struct tag
{
  char * member1;
  char * member2;
} instance =             /* end of declaration part */
{                        /* beginning of initialization part */
  "first",
  "second",
};
```

Listing 1.

```
#define StructBEGIN(Tag) struct Tag {
#define DCL(TypeName)    TypeName;
#define INIT(InitValue)  /* empty */
#define StructEND(Name)  } Name =

"Macros defining declaration part"

#define StructBEGIN(Tag) {
#define DCL(TypeName)    /* empty */
#define INIT(InitValue)  InitValue,
#define StructEND(Name)  };

"Macros defining initialization part"
```

Listing 2.

```
/* File initstruct.h -- begin. */

#undef StructBEGIN
#undef DCL
#undef INIT
#undef StructEND

#if ! defined Hinitstruct
#  define Hinitstruct
   /* definition part */
#  define StructBEGIN(Tag) struct Tag {
#  define DCL(TypeName)    TypeName;
#  define INIT(InitValue)  /* empty */
#  define StructEND(Name)  } Name =
#else defined Hinitstruct
#  undef  Hinitstruct
   /* initialization part */
#  define StructBEGIN(Tag) {
#  define DCL(TypeName)    /* empty */
#  define INIT(InitValue)  InitValue,
#  define StructEND(Name)  };
#endif Hinitstruct

/* File initstruct.h -- end. */
```

Listing 3.

```
StructBEGIN(tag)
 DCL(char * member1) INIT("first")
 DCL(char * member2) INIT("second")
StructEND(instance)

/* File aStruct.h—end. */
```

The file initstruct.h included just before the specification of the initialized structure must define the appropriate set of macros. When the file aStruct is first included, macros implementing the definition part must be defined. When the file aStruct is included for the second time, macros implementing the initialization part must be defined. Therefore the file initstruct.h must be defined as in Listing 3.

Here we have introduced preprocessor symbol *Hinitstruct*, which must not be defined in any source file other then initstruct.h. This symbol will be undefined for odd inclusions of initstruct.h and defined for even inclusions. If aStruct.h and other include files defining initialized structures are used correctly—that is, always included twice on two consecutive lines—this symbol will always be undefined for the first inclusion and defined for the second.

We have to undefine our macros before redefining them. The C preprocessor issues a warning message when a symbol is redefined.

Declaring structures

To make initstruct.h really useful, we must add the ability to generate declarations of the structure without allocating any storage. Our objective is to keep only one specification of the structure that will be included in all source files that need to know the layout of the structure.

In one source file, typically the one containing the function *main()*, the specification should generate the definition of the structure with storage allocation and initialization. In other source files the specification should generate the declaration of the same structure without allocating any storage. We have arrived at the final form for our *include* file initstruct.h.

We now introduce the new preprocessor symbol *Storage*. It must be undefined in source files where only declaration is to be generated without allocating storage. It must be defined in the source file where definition with storage allocation and initialization is to be generated. If it is defined as:

```
#define Storage /* global */
```

the name of the structure variable will be global, that is, visible to all files composing the program. If it is defined as:

```
#define Storage static
```

the scope of the name will be limited to the source file.

At the very end we have added a small test driver. That part will only be compiled if we specify the compilation option:

```
-D DriverH
```

To compile this file, we have to rename it first into initstruct.c. Keeping the test driver in the same file with the code to be tested makes it easy to do a consistency check after any modifications. We do not have to code a test program each time we make a modification. We keep it together with the code to be tested. It also serves as an example on how to use the macros defined in this file.

Recapitulation

Let us now state the complete set of rules for using macros defined in initstruct.h.

The specification of each initialized structure must reside in a separate file. Only structure specification and comments may be placed in that file. For the sake of an example, the name aStruct.h will be used for such a file.

The first noncomment line in aStruct.h must be:

```
#include "initstruct.h"
```

File initstruct.h contains two alternative definitions for each macro. One set becomes defined on odd-numbered inclusions of the file, the other on even-numbered inclusions. The odd macro definitions generate declarations of structure members; the even macro definitions generate initial values.

The second noncomment line in aStruct.h must be macro

```
StructBEGIN(TagName)
```

The argument *TagName* is optional.

The last noncomment line in aStruct.h must be macro

```
StructEND(VarName)
```

The argument *VarName* is the name of the variable of type structure that is to be defined or declared.

The remaining noncomment lines in aStruct.h must be of the form:

```
DCL(Type Identifier) INIT(Value)
```

Each of these lines defines one member of the structure and its initial value. The argument of the macro *DCL()* (short for *DECLARE*) is identical to the declaration for the structure member without the terminating semicolon. The argument of the macro *INIT()* is the initial value.

In the source file where the structure is to be defined and initialized, preprocessor symbol *Storage* must be defined before including the file aStruct.h two times on two consecutive lines (comment lines may intervene). If the symbol is defined as:

```
#define Storage static
```

the structure will be defined with the attribute static. If the symbol is defined as:

```
#define Storage /* empty */
```

the structure will be global.

In other source files where the structure is to be declared as external, preprocessor symbol *Storage* must not be defined before including the file aStruct.h once.

Generalized problem

Apart from the limited usefulness of macros defined here, the file initstruct.h is an example of using the C preprocessor to

Programmers: This new book shows you how your software can support over 150 video terminals and microcomputers!

How to support the wide variety of video display terminals has long been a problem for programmers. The "cursor up" code for one VDT might well clear the screen on another!

If you have spent time searching for control code sequences then this new book will be a welcome relief. We gathered 146 data sheets to give you a single source for your VDT support questions.

A Programmer's Guide to Video Display Terminals shows you how to clear the screen, position the cursor (with examples!), home the cursor, make seven erasures, turn video attributes on and off, and recognize cursor (arrow) keys for over 150 VDTs! We even include data for many VDTs which are no longer manufactured.

You will find this book an indispensible aid if you are a programmer, software developer, consultant, dealer, OEM, value-added retailer, or just frequently called on to support a variety of VDTs.

We are so sure that this book will eliminate your VDT support problems that we offer a FREE 15-day examination. To receive your copy, return the coupon below.

Examine it free for 15 days. If you are not completely satisfied, return it and owe nothing. Prepaid orders will receive a refund.

15-DAY *FREE* EXAMINATION!

A Programmer's Guide to Video Display Terminals
by David Stephens
Atlantis Publishing Corporation Dept. 203
P.O. Box 59467, Dallas, Texas 75229-1467
ISBN 0-936158-01-8 $30 335 pages, softcover

Atlantis Publishing Corporation Dept. 203
P.O. Box 59467, Dallas, Texas 75229-1467

☐ **YES!** Please send *A Programmer's Guide to Video Display Terminals* for 15 days FREE examination. If I decide to keep the book I will pay $30 plus shipping.

Texas residents add sales tax. Price subject to change. Offer subject to acceptance by Atlantis Publishing Corporation. Foreign buyers remit in US currency, specify method and add shipping.

Name ____________________

Company ____________________

Address ____________________

City, State, Zip ____________________

☐ Check or money order ☐ Bill me.
Publisher pays shipping on prepaid orders. Same return privilege.

☐ MasterCard ☐ Visa Exp. Date ________

Card Number ____________________

X ____________________

SIGN HERE. Credit orders invalid unless signed.

```
/* File macro.h -- begin. */

#if     MacroVariant == 1
#  define Macro(al, ..., aN) c0(1) al cl(1) ... aN cN(1)
#endif MacroVariant == 1
#if     MacroVariant == 2
#  define Macro(al, ..., aN) c0(2) al cl(2) ... aN cN(2)
#endif MacroVariant == 2
...
...
#if     MacroVariant == M
#  define Macro(al, ..., aN) c0(M) al cl(M) ... aN cN(M)
#endif MacroVariant == M

/* File macro.h -- end. */
```

Listing 4.

solve particular instances of the following general problem.

Given a table with an arbitrary number of lines *(L)* and a fixed number *(N)* of constants in each line:

```
Macro( a1(1), . . . , aN(1) )
. . .
Macro( a1(L), . . . , aN(L) )
```

generate a fixed number *(M)* of different tables having the same number of lines as the given table and each line having the form:

```
c0(m) a1(l) c1(m) . . . aN(l) cN(m)
```

where $1 <= m <= M$, $1 <= l <= L$, and parameters *c0, . . . , cN* are constant for all lines of one table but are different in different tables.

A generalized solution is to have *M* definitions for the macro *Macro* defined conditionally, as in Listing 4. Note that this solution is easily applied to even more general problems. Constants *a1, . . . , an*, can be ordered differently in different tables. Different subsets of those constants can be used in different tables. Repetition of a constant within a line of the generated table is also possible.

The solution of the generalized problem can be used to generate different but related initialized structures or arrays in different modules of a program or in related programs. For example, similar constant tables may be needed in several passes of a compiler. They could be defined in one *include* file and different variations of the table can be generated for each pass.

For those who'd enjoy more attention to this subject, I've placed a rather long listing called STRUCT.H on the *COMPUTER LANGUAGE* Bulletin Board Service and on the magazine's account on CompuServe. The listing will give you macros for initializing structures. ■

Alexander Abacus has a B.S. in electrical engineering and is a software consultant for CGA Computer Inc., Cranford, N.J. His software experience includes work for Sperry, IBM, and AT&T Bell Laboratories.

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
- Auto memory management
- Keyboard interpreter
- Word wrap
- Auto scroll
- Highlighting
- Color control
- Overlay and restore
- Plus a library of over 50 building block subroutines

Designed for enhanced 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 T4,
Richford, VT 05476
802-848-7738
Master Card & Visa Accepted
Shipping $2.50
VT residents add 4% tax.

CIRCLE 14 ON READER SERVICE CARD

THE fifth generation language

P R O L O G

Implementing the full Edinburgh Syntax as described by Clockson and Mellish. Recognized by Japan as providing unparalleled opportunity for artificial intelligence.

Applications:

▲The highest level of a heirarchial robotic control system.

▲Machine recognition of natural language.

▲Expert systems and knowledge engineering.

Optional:

▲Special libraries

▲Language extensions

Educational Package

$29.95

(MSDOS)

Highlights of LVM Prolog:

▲Type LVM Prolog makes possible the execution of AI applications previously only possible on a mainframe.

▲Invisible compilation to a semantic network provides the flexibility of the interpreted mode and the speed of a compiler.

▲Virtual memory and a sophisticated cache algorithim limited only by the DOS, typically 2 gigabytes.

▲Syntax superset with many extensions such as pattern specified insertion and deletion, clause indexing, robotic control.

▲Unlimited number of resident and virtual modules, each up to 2 gigabytes in size.

▲ ▲ ▲ ▲ ▲ ▲

▲THE most cost efficient package in the industry.

$500- (MSDOS version) Also available for Xenix, Unix, CP/M68K

ADA automata design assoc.

▲VISA, Mastercard, AMEX

▲call or write for brochure

1570 Arran Way Dresher, PA 19025 **technical:** (215)646-4894 **orders:** (215)355-5400

CIRCLE 1 ON READER SERVICE CARD

HIPPO-C™

The C compilers for the Macintosh™

Level 1:

Powerful, expandable, yet affordable:

- A friendly, integrated environment complete with an editor, full compiler, linker, source-level debugger, tutorial, standard C library, and structure definition files.
- Convenient access to serial ports and sound channels.
- Access to over 400 Toolbox routines.
- Over 200 pages of documentation and many sample programs.
- Upgradable to Hippo-C level 2 for $250.00
- New version 1.2
- $149.95

Level 2:

Professional C development system. Here's what you've been waiting for:

- Allows for the creation of large, stand-alone Macintosh applications.
- Comes with an editor, optimizing C compiler, 68000 assembler, linker, C library, stdio package, full floating-point support (including math and trig functions), and structure definition files.
- Convenient access to over 500 Macintosh Toolbox routines.
- Documentation, many sample programs, and sources to useful programs.
- No royalties or licensing fees.
- You may obtain a non-copy protected disk by signing and returning a form along with a nominal fee to Hippopotamus.
- $399.95.

Available from your local dealer or directly from Hippopotamus Software:

HIPPOPOTAMUS
SOFTWARE, INC.

1250 Oakmead Parkway, Suite 210, Sunnyvale, CA 94086 408/738-1200

Dealer inquiries welcome. We accept credit cards, checks, and money orders. California residents add local sales tax. Please include $10 for shipping and handling. Macintosh is a trademark of Apple Computer, Inc. Hippo-C is a trademark of Hippopotamus Software, Inc. Please allow 1–2 weeks for delivery. Price, availability, and specifications subject to change without notice.

CIRCLE 39 ON READER SERVICE CARD

OVERCOME
FORTRAN and PASCAL
LIMITATIONS WITH

$89 Visa/MC

NO LIMIT

A library of 58 Assembler routines transforms MS FORTRAN and PASCAL plus other 8086/87/88 FORTRANs into the flexible, responsive, complete language needed for the microcomputer environment. Ver 1.0 Features:

EXTENSIVE GRAPHICS	(Get, Put, Paint, Color, Dot, Line, Box, Circle, Ellipse, Large Characters)
FULL SCREEN CONTROL	(Windows, Cursor, Read/Write Screen)
STRING MANIPULATION	(Match, Compare, Concatenate/Extract, Pack, Justify, Zero Fill)
KEYBOARD CONTROL	(Read Key During Execution, String Read)
FILE MANAGEMENT	(Exist?, Rename, Delete)
COMMUNICATIONS	(Set Com Line, Send/Receive, Line/Modem Status)
OTHER FEATURES	(Peek, Poke, Determine Time/Date, Random Numbers, Beep, Clear Screen, OR/AND/XOR/NOT/NEG of Byte/Word, Printer Status)

Ver 2.0 with 92 routines now available.
Directories, Command Line Read, Program Chaining, Interrupt Driven Communications. $129. Upgrade $40.

M | E | F Environmental Inc.
P.O. Box 26537 Austin, Texas 78755
(512) 251-5543 **Outside Texas (800) 562-9700**

CIRCLE 55 ON READER SERVICE CARD

SMALL C FOR IBM-PC

Small-C Compiler Version 2.1 for PC-DOS/MS-DOS
Source Code included for Compiler & Library
New 8086 optimizations
Rich I/O & Standard Library

$40

CBUG SOURCE LEVEL DEBUGGER FOR SMALL C

Break, Trace, and Change variables all on the source level
Source code included

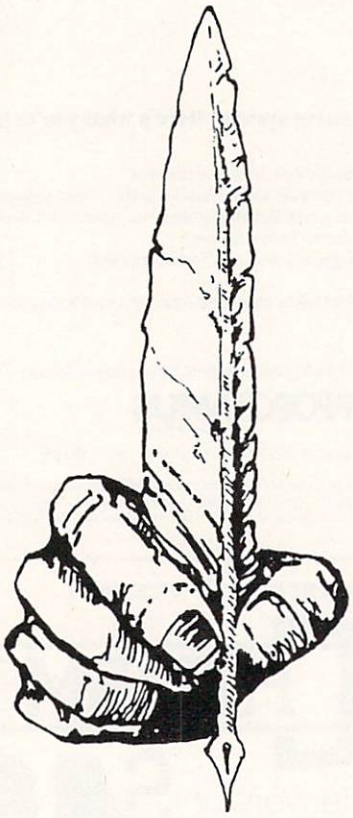
$40

Datalight
11557 8th Ave. N.E.
Seattle, Washington 98125
(206) 367-1803

ASM or MASM is required with compiler.
Include disk size (160k/320k), and DOS version with order.
VISA & MasterCard accepted. Include card no. & expiration date.
Washington state residents include 7.9% sales tax.
IBM-PC & PC-DOS are trademarks of International Business Machines
MS-DOS is a trademark of Microsoft Corporation.

CIRCLE 19 ON READER SERVICE CARD

WHITESMITHS, LTD. 370 COMPILER
NOW YOU CAN RUN BOTH C AND PASCAL ON IBM MAINFRAMES



FEATURES:

- Full implementation of the C programming language for the IBM 370/43XX/30XX
- Supports full ISO Level 0 Pascal, extended to support separate compilation
- Uses standard IBM tools to facilitate debugging in the standard IBM save area layout
- Compatible with the full range of Whitesmiths' compilers (PDP-11, VAX, 68000, 8080, and 8086)
- Provides optional cross support for MS/PC-DOS, CP/M-86, CP/M68k, and CP/M-80
- Runs under all versions of OS (VM, MVS, SVS, MVT, and MFT)
- Includes unlimited use of libraries in binary form
- Runs interactively under TSO and CMS
- Includes first year of technical support

CALL WHITESMITHS NOW AT (800) 225-1030

MASS. RESIDENTS CALL (617) 369-8499

WHITESMITHS, LTD.
97 LOWELL ROAD
CONCORD, MA 01742
TELEX 750246

Distributors: **Australia,** Fawnray Prance Ltd., Rosebery 2018, N.S.W., Sidney 662-4111
Japan, Advanced Data Controls Corp., Chiyoda-ku, Tokyo (03) 263-0383
United Kingdom, Real Time Systems, P.O.B. 70, Douglas Isle of Man, 011-44-0642-26021
Sweden, Unisoft AB, Goteborg, 011-46-31-125810

CIRCLE 43 ON READER SERVICE CARD

PUBLIC DOMAIN SOFTWARE REVIEW

By Tim Parker

A few months ago, another batch of disks from the PC-SIG organization came in the mail. PC-SIG has been mentioned here before as an excellent source of public domain material for MS-DOS and PC-DOS machines. Its catalog now lists 222 disks, covering a wide variety of material. Since PC-SIG's last catalog edition, 24 new volumes have been added.

Probably of most interest to readers is a disk of C utilities that perform a multitude of tasks. PC-SIG vol. 216 contains 41 items ranging from C source to batch files. A documentation file contains details of the volume's contents. A few calendar programs are included for converting Gregorian dates to Julian dates and vice versa, packing date formats (that is, going from 1/2/85 to 01/02/85), determining the day of week from the Gregorian date, returning DOS time and date as integers, and getting time and date in different forms.

The unpacking program is useful when interfacing into packaged programs that expect predefined string formats for the date, and the Julian-Gregorian conversions are really the only accurate method of tracking dates between years. The Julian day is simply a constantly increasing integer, started at an arbitrary date long ago and now reaching into the millions. It does, however, provide an unambiguous method of tracking contiguously throughout time.

A few programmer's utilities are supplied for functions such as reading a string from *STDIN*, writing a string to *STDOUT*, writing a character to the screen using DOS calls, opening menu files and accessing them on the console device, and initializing screen and keyboard arrays. Screen-oriented utilities allow you to erase to the end of the line, toggle screen attributes (blink, reverse, etc.), position the cursor using row-column coordinates, move up or down a specified number of lines, move the cursor on a line by a specified number of columns in either direction, and erase a line from the screen. A pause routine allows timed pauses to be inserted into source code.

Finally, a series of batch files are supplied that emulate UNIX commands, such as *ls*, *cl*, *rm*, *ld*, and others.

Turbo Pascal users will find a utility that acts as an enhanced source lister on PC-SIG vol. 217. It can automatically underline certain words, if requested. The Pascal source code, compiled program, and a documentation file are supplied. A separate list of keywords to be checked is maintained in a data file directly addressable by an editor. A *BLOAD*able screen-dump utility that will work with high-resolution graphic screens is included.

The disk's contents are rounded out with a few more utilities. Probably the most noteworthy is a floppy-disk drive alignment program, supplied with documentation.

An updated version of the popular remote bulletin board system (RBBS) program (version 12.2) is on PC-SIG vol. 212. For anyone interested in establishing a RBBS, this disk supplies all the required software in a tried and tested form. Full help for users is supplied in a series of eight help files describing commands available to the user, supported functions on the system, message protection, message editor, and file menu and subsystems.

Separate programs are supplied for essential BBS operations. The main program, RBBS-PC, is supplied as an .EXE file and a squeezed BASIC version. The unsqueezer is supplied on the disk. A documentation file covers all required aspects of the program. For those with earlier versions of RBBS-PC, update programs convert 12.1 to 12.2 and a list of fixes from version 12.1. A ring-back function is supplied for implementation if the SYSOP desires it. Menus and sign-on prompts are supplied in two forms, one with graphics and one without them. This allows the system to be adapted to different machine configurations without using a debugger.

For anyone contemplating setting up a BBS system, the RBBS-PC software supplies most of the functions that can be required by a starting system operator and surpasses several commercial implementations in many aspects. As these updates show, a continual upgrading policy by the users exists to expand the capabilities of the program.

For telecommunications use, PC-SIG vol. 202 supplies a PC-to-mainframe data communications program. The Simware IBM 3278 version 2.30 program with the keyboard command configuration routines and communication protocol setups does the job that many expensive commercial products do. A CMS/TSO communications program allows error-free use of CMS implementations, as can be found on many university mainframes running IBM or Amdahl machinery and OSVS1 or OSVS2. A start-up batch file gets everything running smoothly. In use, the SIM3278 software performed flawlessly, accessing both CMS and Wylbur systems. For many of the more common commands, definable keys were greatly appreciated.

A couple of new disks cater to word-processing applications. PC-SIG vol. 211 supplies a microjustification routine for left and right microjustification of text. Assembler source code and fairly complete documentation are supplied on the disk. The rest of the disk space is taken up with a speed-reading package.

For WordStar users, PC-SIG vol. 201 has a few WordStar-to-ASCII formatters and their reverse. Two simple word processors are supplied on the disk. Neither will displace a commercial word processor.

And last but by no means least are more utilities. PC-SIG vol. 204 has a useful program called FCOPY that will format and copy single-sided disks in 30 sec. Pascal and object source code are supplied with documentation, although most copy routines are not too difficult to figure out.

A couple of other copy enhancements are on the same volume, including an enhanced delete function. MSPOOL2 allows up to four printer spoolers to be used with the PC. WRTE allows copies of a read-only file to be made.

The next disk in the series, vol. 205, includes an extended batch language with demonstration programs and documen-

Z sets you FREE!

Z — yes! Synergistic combination of ZCPR3 and ZRDOS2 produces flexible state-of-the-art Z80 operating system with tremendous productivity features.

Z-System consists of software modules, dynamic loading segments, and tools permitting optimum computer usage ranging from production program development to turnkey, password-controlled, end-user installations. Facilities include: multiple commands per line, file search paths, named directories, I/O redirection, command flow control, screen-oriented menu generators, complete housekeeping file and directory management, shells, alias (scripts) and nested-alias generation, and complete online help.

Seventy-six support utilities, five tool packages, and two application programs available now! Fully upward compatible with CP/M-80.

Z can now be purchased as auto-install program (Z-Com) or as manual-install ZCPR3 with semi-auto install ZRDOS package (Z-System). Our latest versions, to be released this year, support Zilog Z800 and Hitachi HD62801/64180 high-technology chips, chips run existing 8080 and Z80 programs!

Echelon eight-bit operating systems written in Assembly Language, using linkable macro subroutine libraries, offer performance paralleling best single-user 16/32-bit microcomputer systems.

1. **Z-Com** Full-up Z Operating System with input/output redirection running under CP/M-80, online command and utility documentation and help system **$219.95**

2. **Z-System** Manual-install ZCPR3 and ZRDOS2, easily tailored by programmer to custom needs; source code to core and utilities; similar to Item 1 .... **$199.95**

3. **Z-Tools** Four software development system packages permitting advanced, structured program design, macro relocating assembler, linking loader, librarian, cross-reference generator, debugger, mnemonic and pseudo-op translators, and interactive disassembler. Super $315.00 package value **$200.00**

4. **DSD** Dynamic Screen Debugger offers high-level features never before found in microcomputers; simultaneous display of dual-memory segments, stack, cpu states, and flags, with software In-Circuit-Emulation **$149.00**

5. **The Libraries** Linkable ZCPR3 libraries (Vlib, Z3lib, and Syslib3) of over 400 subroutines used for Assembly Language program writing. Simplifies structured, efficient code production; online help system and full source code provided **$45.00**
Syslib3 alone **$29.00**

6. **Term3** New generation communication program permits menu control of computer/modem operations between operator and time-share services, bulletin-boards and other remote computer systems; auto-answer to command-line prompt **$99.00**

7. **Discat** Fancy file and disk catalog program running under Z-System, menu driven and easily customized by operator **$49.00**

Fortnighter newsletter, 24-hour BBS Z-Node System keep Z users informed of microcomputer happenings. Write or call for brochure or order now! State disk format desired; add $3.00 shipping & handling; Californians please add 6-1/2% sales tax. Visa/MC, check, money or purchase order accepted. (Program names are trademarks of their respective owners.)

Echelon, Inc.

101 First Street • Los Altos, California 94022 • 415/948-3820

CIRCLE 22 ON READER SERVICE CARD

tation. A program called CURSOR allows the cursor's shape to be changed, while CV2 changes diskette labels with DOS 2.0.

NDOSEDIT is a DOS command editor with a command stack, resembling VM's editor. CMS's browse command is simulated by another routine. Lastly, CGCLOCK2 displays a color clock that beeps every 15 min. The excitement never ends!

It would be impossible for any programmer keeping abreast of the trends in this market to ignore the overwhelming growth of the C language. After the February issue of *COMPUTER LANGUAGE*, which was devoted almost exclusively to C, a number of readers asked me where versions of C can be found in the public domain.

I have mentioned the Small-C implementation of the popular language in this column before. As noted earlier, the version is not a full Kernighan and Ritchie implementation but does contain enough of the flavor of C to allow a newcomer to the language to get the feel of it. Small-C allows a programmer to develop fairly complete program tasks without shelling out several hundred dollars for a commercial C compiler.

Small-C is not without its problems. Virtually no documentation exists for the system other than text files bunched together. This is overcome by the wealth of good books available on C (and everyone who uses C has to have a Kernighan and Ritchie textbook).

Secondly, the inherent limitations of the implementation may frustrate some. However, by the time those limitations are reached, the user should be ready to move up to a full-blown C version.

Hence, Small-C provides a very easy entry vehicle into the language of the eighties for many programmers. Luckily, there are versions of Small-C for almost every operating system. Most bulletin boards now stock the most popular of these for CP/M 80, and they can be obtained from several of the usual public domain software sources. Addresses are appended for further information.

The version that seems the hardest to locate is the one for CP/M 86. However, CP/M 86 has Small-C available in several sources, including SIG/M (Special Interest Group/Microcomputers) vol. 149. Small-C is supplied as a library file with a small documentation file. This version is also now available on the *COMPUTER LANGUAGE* BBS in case you can't track it down anywhere else.

PC-SIG is at 1556 Halford Ave. Suite 130, Santa Clara, Calif. 95051. SIG/M is at P.O. Box 2085, Clifton, N.J. 07015-2085. ■

EXOTIC LANGUAGE OF THE MONTH CLUB

APT: Automatic and Programmable Trees

By Peter Reece

People generally agree that certain basic elements are desirable in all languages. Items like string manipulation and a good set of mathematical functions are considered essential. Less obvious considerations such as good readability and structured code are gradually receiving the same status.

Beyond these basics, however, the designer of a general purpose language has no way of knowing what tasks the user will be asking the language to perform. Similarly, the designer cannot predict the power of the computer on which the language will be used. It therefore makes sense to provide tools which enable a programmer to add to the basic commands which were originally built into the language. Ideally, the language should get smarter as it is used. A user should be able to tailor it to the tasks for which it is most often used and to the available computer resources.

Automatic and Programmable Trees (APT) is such a language. It contains a highly robust command set and a complex syntax while at the same time having a low-resource overhead (without sacrificing speed, efficiency, or ease of use). In addition, APT is easily customized. It can grow with the user, both automatically extending itself as programs are written and providing the means for generic extensibility—of creating commands which create commands.

This is further enhanced by the provision of several features found only in some special purpose languages, such as LISP-like lists, stacks, toolbox structures, LOGO graphics, self-modifying code, etc. Figure 1 depicts an overall model of the language. This article discusses some of the ways in which APT accomplishes these functions.

We'll begin by using some simple programming examples to study the language's ability to grow. Executable strings, toolbox files, hierarchical workspaces, and trees will all be discussed in this context.

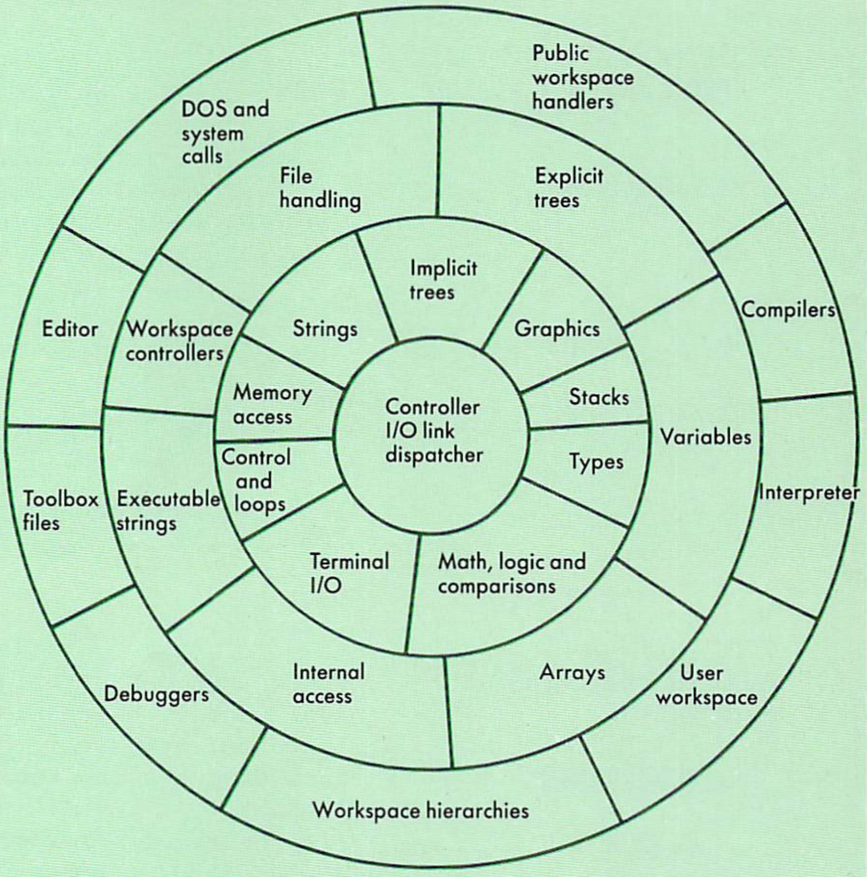
Next we'll look at APT's internal structure in which trees, threads, and optimization techniques contribute to the overall power and flexibility of the language. Finally, we'll take a look at APT in the total context of present day languages and language design strategies in general.

Like LISP or Forth, APT is an extensible language. This term has a number of meanings but generally refers to the ability of a language to grow—to add to itself or to be added to and customized without any changes to its basic command set. In APT this can occur in several ways, including procedures, toolbox files, builds, trees, workspaces, Assembler, and executable strings.

Like COBOL or Pascal, APT is a structured language composed of one or more modules called procedures. These are arranged in a hierarchical bottom-up manner, that is, the most primitive operations are defined first. APT procedures are invoked by name and, once defined, may be used in a number of ways (Listing 1).

As you can see, once a procedure is defined, it can be used as if it were a command, an argument, a variable, data, or in fact any other valid APT construct. Like LISP, APT does not recognize differences between data or programs unless specifically told to do so. Therefore any expression may be substituted for any other, providing, of course, that the context is

A model of APT and its components



The major internal divisions in APT are shown. Distance from the center indicates decreasing complexity in the internal design of the language—e.g., the complexities involved in designing implicit trees were greater than those required to design explicit trees. The various features in the diagram are elaborated in the accompanying text.

Figure 1.

appropriate. Consequently, defining a procedure automatically adds a new function to the language. As we shall see, this allows for considerable flexibility in programming.

APT may also grow through the use of executable strings. Any program may execute or compile any string or string array. Because the language provides a good set of string-manipulation tools it is a simple matter to modify then execute strings from within a running program. Hence a program can write another program, compile it and execute it all without programmer intervention. A trivial example of this is seen in Listing 2. The *BUILD* statement in APT is conceptually similar to the statement in Forth or STOIC of the same name. Its complete format is:

BUILD Name . . . WHENDEF . . .
WHENRUN . . . END

The three dots (. . .) represent any valid APT expressions. Basically, *BUILD* creates a command by executing the statements between *WHENDEF* and *WHEN-RUN*. When this new command is invoked, it will execute the statements

```
1. PROC EXAMPLE           ; Define a procedure named EXAMPLE (APT
                          ; automatically numbers each line in a procedure).
2. LOG(X+2.4)/3.1         ; Find the natural logarithm of the
                          ; expression, then divide it by 3.1

3. END                    ; End the procedure definition

EXAMPLE                   ; Typing EXAMPLE will execute the
                          ; procedure - EXAMPLE here is used as a command.
COS(EXAMPLE)              ; Find the cosine of log(x+2.4)/3.1 -
                          ; here EXAMPLE is used as a argument to a function.
EXAMPLE+2.1 -> TEST       ; Here EXAMPLE is used as a variable. It
                          ; is added to 2.1 and the result is saved in TEST.
```

Listing 1.

```
1. PROC EXECUTABLE.STRING.EXAMPLE
2.    "PROC ABC 2+3->I END" -> DUMMY  ; Place the string (in
                                      ; quotes) into the string
                                      ; variable DUMMY.
3.    EXECUTE(DUMMY)                  ; Executing DUMMY creates
                                      ; the procedure called ABC.
4.    ABC                             ; This places 5 into I
                                      ; (i.e. the just created
                                      ; proc ABC will execute.)
5.    "XYZ"->DUMMY(6:3)               ; This replaces the 'ABC'
                                      ; in line 2 with 'XYZ'.
6.    EXECUTE(DUMMY)                  ; Executing DUMMY now
                                      ; creates a new procedure
                                      ; - XYZ - in addition to ABC.
7. END                                ; End the procedure definition.
```

Listing 2.

```
1.  PROC TOOLBOX.EXAMPLE      ; Procedure demonstrating simple toolbox calls
2.       2+3->I               ; Add 2 and 3, place the result
                              ; into the variable I.
3.       RUN"AVERAGE"         ; This executes the toolbox program
                              ; AVERAGE - AVERAGE is used as a simple command
4.       (I+RUN"VARIANCE")/2  ; I + the result of VARIANCE is
                              ; divided by 2 - here a toolbox
                              ; file is used as a variable
5.  END                       ; End the program
```

Listing 3.

between *WHENRUN* and *END*. Hence the *BUILD* construct is used primarily to create a class of commands which creates other commands which in turn share a common run-time code—the *WHENRUN* portion. This is clarified in Figure 2.

Like the C language, APT allows the programmer to build and compile a set of utility, or toolbox, files which may be included in a program at will. Two options are available. First, the editor (more about APT's editor later) enables any number of source programs to be brought in from disk and merged into a user program.

A more powerful technique, however, is the inclusion of a *RUN "XXX"* statement in a program, where *XXX* is the name of a previously compiled program. APT will automatically execute *XXX* if it is currently in memory, or automatically load it from disk prior to execution if it is not in memory.

Like any other APT function, toolbox files may also be used interchangeably as variables, commands, data, or arguments. Listing 3 is an example. As you can see, the programmer can build a set of toolbox files, save them in compiled format (that is, object rather than source code) and include them at will in a program. Variables may be shared by toolbox programs since, like FORTRAN, APT allows the creation of a COMMON block through

BUILDing extensibility

The instruction:

```
BUILD ABC WHENDEF ...
          WHENRUN ...
END
```

does the following:

1. BUILD—places 'ABC' into the current dictionary
   —instructs the compiler that ABC is to be used to define commands, i.e., it is a command that builds other commands
2. WHENDEF—all statements between here and WHENRUN will be executed when ABC is used in a program to define another command
3. WHENRUN—all following statements will be executed when the command created by ABC is run

For example:

```
BUILD ABC WHENDEF COMPNUM
          WHENRUN + ARGS ?
END
```

COMPNUM—compiles the number on the stack into memory
ARGS—gets arguments from the input stream
+ —adds two numbers
? —prints the value of the top entry on the stack
Hence, if we write:

2 ABC "NEWCOMMAND"

then NEWCOMMAND will be entered into the current dictionary. When NEWCOMMAND is used in a program, the number 2 will be placed onto the stack. If we now write:

NEWCOMMAND (5)

then 7 will be printed – 2 is placed onto the stack, 5 is read from the input stream (ARGS), 2 and 5 are added (+), and the result is printed (?).
Similarly, 7 ABC "ANOTHER.COMMAND"

ANOTHER.COMMAND(10) will print 17.

This works as follows:

Command	**Points to**	**Which contains**
ABC	whendef code for ABC	2
NEWCOMMAND	whenrun code for ABC	+, ARGS, ?
ANOTHER.COMMAND	whenrun code for ABC	+, ARGS, ?

Figure 2.

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

```
PROC TRIVIAL.TREE.RESTRUCTURING.EXAMPLE
     TREE("A","B","C","D")  A        ; Create a tree, then execute it.
     TREE("A","C","B","D")  A        ; Change it, then execute it again.
END
```

Listing 4.

```
CREATEWS MEDICAL(5000)  ; Create a medical workspace and
                        ; allocate 5K bytes to its use.
CHANGEWS(MEDICAL)       ; Enter the newly created workspace
EDIT                    ; Summon the editor to begin creating
                        ; rules involving medicine onto the workspace.
```

Listing 5.

which variables may be passed when using overlaying (placing one program on top of another in memory).

If the program AVERAGE, in line three of the procedure *TOOLBOX.EXAMPLE*, contained a command *RUN"AVERAGE"*, then the program would call itself (it would already be resident in memory). Toolbox files may therefore be used recursively. All this means that toolbox programs may be brought in from disk, be executed, call another program, and pass information to one another without user intervention.

A very important construct in APT is the tree structure. Trees in APT are similar to lists in LISP. Lists of procedures may be linked or unlinked without programmer intervention. Suppose that we have four procedures called A, B, C, and D. The command:

```
TREE("A","C","B","D")
```

will form a simple connection between these separate procedures. This connection is called a tree, in this case named A, which looks like this:

```
A —> C —> B —> D
```

Each element of the tree is termed a node. When A is invoked, each node of the tree will execute in turn—that is, node A, then node C, B, and finally node D. Exactly the same thing could have been accomplished by creating a procedure as presented below:

```
PROC EXAMPLE
A C B D
END
```

When *EXAMPLE* is run, A, C, B, then D will execute. Why create a tree if a procedure will produce the same result? Because a tree may be altered by a user, a program, or another tree at will (Listing 4).

DEZIGN

THE STRUCTURED PROGRAMMING TOOL FOR MODERN TIMES

- Design your programs right on the screen, using modern techniques based on the popular Jackson Structured Programming method (JSP)!
- **DEZIGN** is more than just another flowcharting tool. It is an integrated tool for designing and documenting programs and for generating ADA, C, PASCAL, and PL/I source code, as well as dBASE II and dBASE III command files.
- **DEZIGN** enables you to create Data and Program Structure Diagrams using the Sequence, Selection (IF-THEN-ELSE), and Iteration (DO WHILE) constructs; assign detailed statements to the diagrams; and synthesize source code from the control logic represented on the diagrams and the detailed statements assigned to them.
- **DEZIGN** lists for $200. It runs on the IBM PC, XT, or AT and requires 128K RAM, one disk drive, and an 80-column color or monochrome display.
  - DEZIGN-PC runs under DOS 2.0, 2.1, and 3.0.
  - DEZIGN-86 runs under CP/M-86 1.1.
- Want to learn more? Please contact us concerning pricing and availability of JSP reference texts and seminars.

ZEDUCOMP • P.O. BOX 68 • STIRLING, NJ 07980

(201) 755-2262

dBASE II and dBASE III are trademarks of Ashton-Tate, Inc.

CIRCLE 83 ON READER SERVICE CARD

To pursue the tree analogy further, techniques can easily be developed in APT for adding new branches to trees. Trees may be pruned, nodes may be extended, entire trees may be displayed and so on. Commands also exist for pattern matching of trees, selecting branches based on template matching or mismatching and similar functions.

All nodes are either procedures or other trees. Thus a given node may contain any valid APT command, including tree-manipulation commands. Trees may therefore self-modify, allowing the language to grow as trees grow.

Once a tree growing program has executed and a given tree has been constructed, a single command, *TRACETREE*, may be issued to find out what nodes exist and in what order in the final tree.

While beyond the scope of this article, APT's tree structure is ideal for certain types of data base creation. For the reader familiar with data base design, imagine that tree nodes represent sets containing either subsets, data arrays, or data files with schemas and subschemas being embodied in tree definitions.

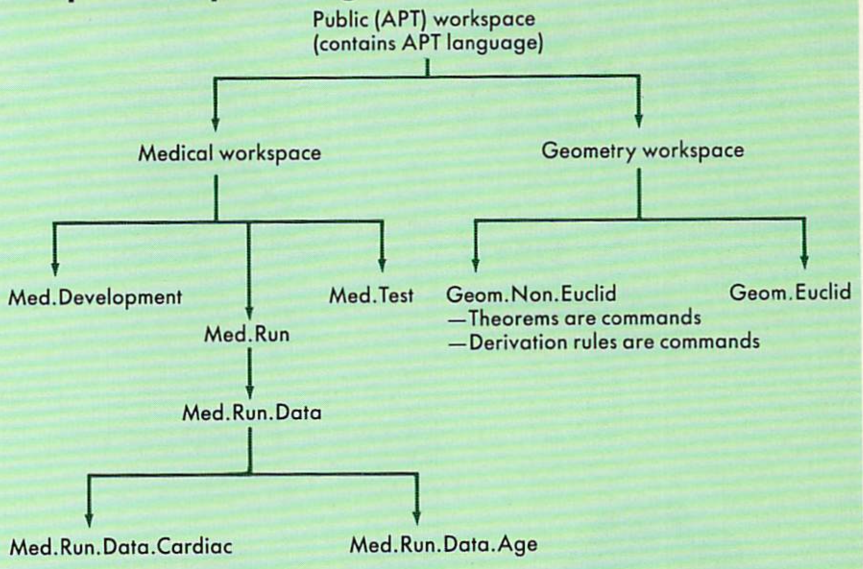
Tree commands may then be used to rapidly navigate through the data base to reach the data in a given set. Trees may be manipulated as wholes, broken apart, and reassembled into new structures at will. This gives a tremendous flexibility to the language, allowing it to be used in applications as diverse as artificial intelligence, game playing (for example, chess) or data base design without sacrificing ease of use.

Like expert systems, (for example, Solver or Dendril), APT enables a user to create data and rules for manipulating that data for a particular class of problems. These rules and data may be isolated from those pertaining to other problem classes by means of a logical construct called a workspace (Listing 5).

Any number of workspaces may exist. Where appropriate, they may even be logically grouped together into workspace hierarchies resembling a tree structure. Figure 3 illustrates how, to a medical user, APT could appear to be a medical programming language with several levels, while to a mathematical user, it could appear as a mathematical language, depending upon which workspace hierarchy is in use.

Each workspace contains its own list of reserved words called a dictionary. Dictionaries contain the names of all functions in the workspace and pointers to the code for these functions (Figure 4). As such, when a new problem-specific dictionary is created within a workspace, a metalanguage results which uses APT's reserved words as a subset.

Sample workspace usage



All of these workspaces exist within the public workspace. To create the medical workspace, the user would (while within the public workspace) type:
'CREATEWS MEDICAL (size)'
where 'size' is the maximum amount of room to be allocated to the workspace. Med.Run would be created from within the MEDICAL workspace, not the public workspace.

Med.Run.Data users can access the Med.run.data.xxx workspaces, but not the Med.Development, Med.Run, Med.Test workspaces. No MEDICAL users can access GEOMETRY, and vice versa.

Conceptually, APT's workspace structure resembles the layout of an account structure on a VAX or PRIME computer, for readers familiar with these systems.

Figure 3.

Dictionary structure

This example shows a dictionary entry for the APT command 'RUN' (which executes a program from disk).

	Dictionary Entry	**Action**
1st byte:	Type byte	—Indicates the type of command (variable, command, proc, array, etc.)
		—Indicates the internal level (i.e., execute only, compile only, requires more parts, etc.)
		—Indicates whether the command takes arguments
2nd byte	Usertype and internal code	—Indicates whether or not the token RUN is strongly typed, and if so, what its type is
		—Used to flag special compiling options
3rd-5th byte	R U N	—As many bytes as are necessary to hold the name of a command occur from byte #3 onward
6-7th byte	Address of RUN routine	—Compiler compiles a routine branch (two bytes) to this address into the program object code
		—Interpreter branches to this address if current mode is interpret mode

Note: APT may have any number of logically isolated dictionaries. Dictionaries are only used by the compiler, interpreter, and debugger—they are not in memory during program execution.

Figure 4.

Does your **ISAM** run on **IBM, APPLE, DEC** and **AT&T** computers? **c-tree** does, and you only **BUY IT ONCE!**

c-tree™
BY FAIRCOM

2606 Johnson Drive
Columbia MO 65203

The company that introduced micros to B+ Trees in 1979 and created ACCESS MANAGER™ for Digital Research, now redefines the market for high performance, B+ Tree based file handlers. With c-tree™ you get:

- complete C source code written to K&R standards of portability
- high level, multi-key ISAM routines and low level B+ Tree functions
- routines that work with single-user and network systems
- no royalties on application programs

$395 COMPLETE

Specify format:
5¼" PC-DOS 3½" Mac
8" CP/M® 8" RT-11

for VISA, MC or COD orders, call
1-314-445-6833

Access Manager and CP/M are trademarks of Digital Research, Inc. Apple is a trademark of Apple Computer, Inc. c-tree and the circular disc logo are trademarks of FairCom IBM is a trademark of International Business Machines Corporation DEC is a trademark of Digital Equipment Corporation ©1984 FairCom

CIRCLE 26 ON READER SERVICE CARD

ADVANCED PROCESSORS ARE OK
TUNED SOFTWARE IS BETTER

Performance. Every user wants more of it.

Advanced hardware is one solution. But advanced hardware is difficult to obtain and expensive. Meanwhile, there are MILLIONS of existing machines waiting for a new generation of faster, more robust software. How will your software measure up?

Performance optimization should be considered part of the software development cycle.

Many programs spend 90% of the time executing less than 10% of the code. A small amount of time invested in optimizing these busy areas can yield big performance dividends. But rarely is it obvious just what pieces of source code account for 90% of the processor time. You need a sieve that can quickly isolate the big pieces (timewise) of your program.

CODE SIFTER is the tool you need to identify the time consuming sections of your EXE or COM file.

It observes your running program and generates a statistical report that indicates which areas of your program are prime candidates for your optimization efforts. The statistical output includes the symbols from your LINK map and can be directed to your display, printer, or to a file.

CODE SIFTER has advanced features.

Like a user friendly interface with function keys and online help. An adjustable sampling rate provides more accurate statistical data. An iteration option can rerun your program a number of times to reduce the busy areas to very narrow ranges. A demonstration program (with source) is provided for use with the step by step tutorial in the manual so that you can become productive in just a few minutes. CODE SIFTER works with any source language and does NOT require a knowledge of assembler to use. And if you are still not convinced – we will loan you a demonstration version.

David Smith Software
Box 25A R.D.#3
Oxford, N.Y. 13830
(607) 843–6209

$119

Requires
IBM–PC or XT
128K DOS 2.x

IBM is a registered trademark of International Business Machines Corporation

CIRCLE 36 ON READER SERVICE CARD

In this sense any number of these metalanguages may be built so that each specialized user may program using terminology with which he or she is familiar. A medical programmer would use medical terms; a mathematics programmer would use mathematical terms.

Dictionaries are used during program development only. Unless specifically told otherwise, APT eliminates a dictionary from a workspace during program execution in order to conserve memory.

Once a given workspace or hierarchy has been set up, subsequent programming becomes very simple. The user may add to the current dictionary at any time, creating classes of commands peculiar to a certain type of problem.

Since each workspace contains its own dictionary, rules for one type of task are logically isolated from those commands better suited to other problem classes. A dictionary in a hierarchy is accessible to those workspaces higher than it in the same hierarchy. The same structure can also be used to isolate data (for example, test data from live data—Figure 3, Med.Test vs. Med.Run.Data), or even different users from one another.

To isolate users, an interrupt clock could switch between workspaces in a circular queue. A user would be assigned a workspace which would be active for *n* ticks of the clock, at which time the next user workspace in the queue would become active and so on. (Saving a given user's current status is very simple since, internally, all workspaces operate via a special stack architecture). Thus APT's workspace strategy also lends itself to time-share activity.

An important additional point is that the use of workspaces allows the language to maintain its integrity. Although the workspace structure allows APT to grow in meaningful ways, it cannot degenerate into different dialects.

The public workspace, the language's central repository of commands, is available to all other workspaces and is itself never altered. Different hardware, for example, would see the addition of a workspace specific to that hardware. The central public workspace would not be altered. In this way, regardless of the APT environment, metalanguage or computer with which a programmer is working, the basic command set will always be the same. The dialectic confusion typical of BASIC or FORTRAN, for example, would not occur.

The need for a programmer to resort to machine language is practically eliminated by APT. This is because the language contains a number of commands specifically for the control of I/O ports as well as commands which can peek and poke memory both on a byte-by-byte or block-by-block basis. Commands also

```
ASSEM ADD               ; This assembler procedure will use hardware
                        ; registers to add two integers.
      POP HL  POP DE    ; Z80 code - pop the top 2 stack entries into
                        ; the hl, then de registers.
      ADDR(HL,DE)       ; Add hl and de, leave the result in hl.
      PUSH HL           ; Push the result onto the stack.
ENDASM                  ; End the assembler procedure.

PROC EXAMPLE            ; This procedure demonstrates how the
                        ; assembler routine may now be used just like
                        ; any other.
     2 4 ADD            ; Use the ADD routine to sum the top two
                        ; numbers on the stack - in this case 2&4
END                     ; End the procedure.
```
Listing 6.

exist for jumping to and returning from non-APT programs which reside in memory.

Sometimes, however, a specialized task requires direct access to machine language. To allow this to happen as painlessly as possible, APT is equipped with an assembler. By adding special assembler procedures to a program, it is a simple matter to extend APT to take advantage of particular hardware. Assembler commands make use of a special type of procedure called the *ASSEM* procedure (Listing 6).

Various assembler functions exist. For example, assembler code in APT is structured code—*FOR . . . NEXT*, *BEGIN . . . END*, etc., may be used. Also an assembler procedure may *CALL* other assembler procedures by name. Finally, APT's assembler provides easy methods of hooking into other non-APT programs.

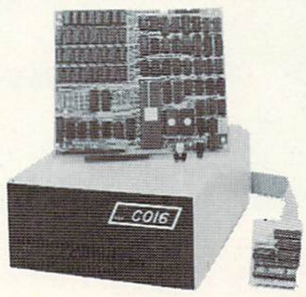
Almost all programming in APT is done through APT's editor. This is a front-end interactive system through which a programmer writes, compiles, tests, debugs, and executes a program. When writing a program the editor is used pretty much like a word processor. Full-screen scrolling, wildcard searches, split screens, block moves, formatted printing of source text, merging of files from disk, DOS access, and various other word-processing features are all available with a single keystroke.

Prompts and help messages appear where appropriate. When a program has been written, a single keystroke causes it to be compiled. The area of the buffer to be compiled lies between the cursor and a *.STOP.* statement so that the entire program or only one or more procedures may be selected for compilation. This is very handy during program development.

If an error occurs during compilation,

A POWERFUL 68000 DEVELOPMENT ENVIRONMENT FOR YOUR Z80 SYSTEM

CO1668 ATTACHED RESOURCE PROCESSOR



- 68000 Assembler
- C Compiler
- Forth
- Fortran 77

- Pascal
- BASIC-PLUS
- CBASIC
- APL. 68000

6 MHZ 68000   CP/M-68K   768K RAM
4 x 16081 MATH CO-PROCESSORS   CPM80 RAM DISK

Develop exciting 68000 applications on your current Z80 based CPM system using powerful mini-frame like 32 bit programming languages. And then, execute them at speeds that will shame many $100K plus *mini*computer systems.

The **CO1668 ATTACHED RESOURCE PROCESSOR** offers a Z80 CPM system owner a very low cost and logical approach to 68000 development. You have already spent a small fortune on 8 bit diskette drives, terminals, printers, cards cages, power supplies, software, etc. The CO1668 will allow you to enjoy the vastly more powerful 68000 processing environment, while preserving that investment.

CO1668 ATTACHED RESOURCE PROCESSOR SPECIAL FEATURES:

- 68000 running at 6 Mhz
- 256K to 768K RAM (user partitioned between CPU and RAM Disk usage)
- Up to four 16081 math co-processors
- Real time clock, 8 level interrupt controller & proprietory I/O bus
- Available in tabletop cabinet
- Delivered w/ sources , logics, & monolithic program development software
- Easily installed on *ANY* Z80 CPM system
- CP/M-68K and DRI's *new* UNIX V7 compatible C complier (w/ floating point math) - standard feature
- *Can be used as 768K CPM80 RAM Disk*
- Optional Memory parity
- No programming or hardware design required for installation
- Optional 12 month warrantee

- PRICES START AS LOW AS $899.00 for a CO1668 with 256K RAM, CPM68K, C Complier, Sources, Prints, 200 page User Manual, Z80 Interface, and 68000 System Development Software.

For further information about this revolutionary product or our Intel 8086 Co-Processor, please send $1 (no checks please) or call:

HSC INC.

Hallock Systems Company, Inc.
262 East Main Street
Frankfort, New York 13340
(315) 895-7426

RESELLER AND OEM INQUIRIES INVITED.

CIRCLE 31 ON READER SERVICE CARD

```
PROC EXAMPLE.OF.ERROR.TRAPPING   ; After this routine has executed,
                                 ; errors will be trapped automatically.
     ERROR(XYZ)                  ; If an error occurs, the user procedure
                                 ; called XYZ will execute.
     BREAKOFF                    ; Disable escape keys - i.e the user
                                 ; cannot exit a program illegally.
END                              ; End the procedure.
```

Listing 7.

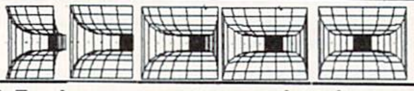
```
A BECOMES B         ; All references to array A now refer to array B
A BECOMES ITSELF    ; All references to array A now refer to A
```

Listing 8.

```
INTEGER APPLES     INTEGER ORANGES           ; Create two integer variables
APPLES->ORANGES                              ; This is legal since
                                             ; the only type attribute of the
                                             ; variables is integer.
SETYPE("APPLES",1)  SETYPE("ORANGES",2)      ; Arbitrarily choose a
                                             ; strong typing of '1'
                                             ; for APPLES, '2' for ORANGES.
APPLES+I->APPLES  ORANGES+I->ORANGES         ; Quite legal - I is type integer.
APPLES->ORANGES                              ; Illegal - APT displays
                                             ; TYPE MISMATCH even
                                             ; though both variables are integer.
```

Listing 9.

What else could you do, if you had a
3-D GRAPHICS LANGUAGE to harness the
computational power of your computer?



E.G. is a new programming language that
provides most of the computing power of
a dedicated CAD SYSTEM at an incredible
price: less than one cent on the dollar

E.G. is a complete graphics language:
* automate the generation of drawings
* design math intensive products
* unwrap & template curved surfaces
* output ascii files to drive n.c.tools
* accuracy exceeds 1 part in 32,000

43 commands control looping, branching
disc access, internal data structures,
subroutines and output devices.

E.G. is also a powerful post-processor:
* plot your numerical data in 3-D
* virtually unlimited macro capability
* write "turn-key" applications
* 1000 data points, 1000 lines per file
* optional "best fit" plot algorithm

Easy to use, rather than easy to learn

E.G. provides the programmer with every
mathematical operation & function found
in BASIC + powerful, proprietary shape
generating and manipulating commands.

EUCLID GRAPHICS

SYSTEM: 64K APPLE D.O.S 3.3

developed at
SEVEN SEAS SOFTWARE
35 Cape George Wye
Port Towensend, Wa.
98368
(206) 385-3771

MANUAL $ 25
COMPLETE $200
free updates
for one year

signed nondisclosure agreement required

CIRCLE 28 ON READER SERVICE CARD

the source debugger for lattice C

Your time and convenience come first! The MSD C Debugger™ is the last, and perhaps final, word in programming assistance for Lattice C users. C Debugger produces a high level view of C programs via function names, line numbers, variable names and C data types, plus a low-level view of machine addresses and instructions for testing assembler language functions.

More features include:
- All documentation is prepared for programmers.
- Online help screen throughout the process.
- Capability to single step through your program.
- Set break points, examine registers and variables.

$165.00 + $3.50 shipping

VISA MasterCard

MSD

To order, call or write:
MICRO-SOFTWARE DEVELOPERS, INC.
214½ W. Main St. • St. Charles, IL 60174
312/377-5151

Lattice C is a trademark of Lattice, Inc.

CIRCLE 54 ON READER SERVICE CARD

the editor will position the cursor over the error and print the nature of the problem at the bottom of the screen. The user then may repair the error and hit a single key to continue compilation. It is not necessary to restart compiling from the beginning of the source text.

Once the program has been compiled without error, it may be executed by the editor, again with a single keystroke. During development the programmer may execute either the entire program or a procedure(s) from the editor. When the program completes execution, or an error occurs, the editor will again resume control. It may also be used to feed values to variables in procedures for testing purposes, to place data into arrays or onto the stack or, in fact, to access any APT command at all.

All of this makes the actual mechanics of programming, compiling, debugging, and executing very simple and straightforward. Debugging is particularly simple.

From the editor a wide variety of debugging techniques are available. In addition to partial compilation and testing of individual procedures and program parts, debugging control of a running program is possible.

Programs can be single stepped with each step scrolling horizontally across the bottom of the screen, freeing the rest of the display for use by the program. During single stepping, loops may be automatically executed at full speed if desired. Breakpoints may be set. That is, a program may be run until a chosen procedure or command is encountered. Execution will then halt and instructions may be given to continue, print the value of a variable, skip a loop and so on.

Programs may also be traced while operating at full speed (program steps are printed on the last line of the screen). Decompilation of compiled code is also possible, although this requires that the current workspace contains a dictionary corresponding to the program being decompiled. Recall that dictionaries are used during development only; they are not needed while a program is running. Finally, error trapping within a program is easily accomplished (Listing 7).

The dictionary in the public workspace contains literally hundreds of commands. While it is impossible to describe all of them here, some of the more interesting ones may help give you a better feel for what tools are available to the programmer. For example, commands exist to create and modify arrays at run time (that is, after compilation). An array may even be made equivalent to another array by a program while it is running. That is, all references to array *A* may be made to refer instead to array *B* without destroying any data in the original array *A*. When desired, references to array *A* may be made to refer again to *A* (Listing 8).

Another powerful command is the *PLOT* statement, which is part of APT's graphics package:

```
PLOT("TIME","FREQUENCY","SPEC
    TRUM",ABC(1,1))
```

will automatically scale and plot the data starting at the first cell of array *ABC*, onto a plot which will be labeled SPECTRUM with an x-axis labeled TIME and a y-axis labeled FREQUENCY.

APT also allows the programmer full control of variable, array, and even command type designations. For example, variables may be weakly or strongly typed (Listing 9). Infix (normal algebraic) notation and postfix (stack) notation may be used interchangeably in APT. The following two expressions, for example, yield identical results:

```
(2+5)/(17*2)
2  5 STACK+ 17  2  STACK+ STACK/
```

There are a large number of stack-manipulation commands in APT. These are useful when variable storage space is at a premium. Stacks do not use memory at the same rate as variables. Also, passing recursive arguments is very simple with stacks.

All procedures in APT are recursive. Even toolbox files call themselves recursively. The two procedures in Listing 10, which produce identical results, illustrate both recursion and passing arguments using variables and stacks.

Other facilities of interest include those presented in Table 1.

Having touched briefly on some basics of APT, it may be useful to step back a moment and take a look at a longer program than we have discussed so far.

On the *COMPUTER LANGUAGE* BBS and on this magazine's account on CompuServe, I've put a copy of a program called Conway's Game of Life which was too long to be printed in the magazine (look for the file name APT.LTG in the April listings). It demonstrates an unsophisticated APT program to play the popular Game of Life. This should give you a feel for what a typical program looks like.

As you read through the program and the comments, notice the high level of readability which the language offers and the straightforward syntax. In general, the language is very forgiving of syntactical errors since the rules are few and often optional. Points to note are the mixing of stack and algebraic notations (for example, in the second procedure), the use of the *SELECT* statement (one of several variants of APT's case statements), and the simple manner in which large amounts of data may be passed between I/O and an array (the *MOVE* statement).

The way that APT accomplishes all of this is surprisingly simple. Internally APT operates in three distinct ways: threads, trees, and optimization.

First, I'll talk about threads.

A compiled program is really a series of branches to previously defined procedures or commands. Since the addresses of these procedures are usually scattered throughout memory, some method must exist to keep track of where the branch originated. Otherwise, there would be no way of returning to a user's

APT facilities

Sort	To sort arrays, data, or memory locations
Graphics	Commands for LOGO turtle graphics, cartesian, polar, and normal coordinate systems, vector graphics, hi/low- resolution drawing, automatic plotting and scaling, etc.
Pointers	Similar to pointers in the C language; access to the absolute and/or relative address
Mathematics	A full set of functions is available (sine, log, etc.), including complex numbers.
DOS calls	Calls to the operating system are easily accomplished with a single command.
Files	Many commands e.g., any number of files may be open at a time, random access, mixed types, etc.
Control	A whole set of loop and control functions (While, Begin, For, Case, etc.)

Table 1.

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

Operation of the thread dispatcher

Suppose that procedures A,B,C, and D exist and occupy the following contiguous locations in memory:

B	location 7AFC to 7B2F
D	location 7B31 to 8190
A	location 8192 to 917C
C	location 917E to 9180

If a procedure 'Q' is compiled beginning at location A100 as:

```
PROC Q
     A B C D
END
```

then when Q is executed:

Address on dispatch stack	Address pointed to by dispatch stack	Routine executed
A100	8192	A
A102	7AFC	B
A104	717E	C
A106	7B31	D

This would result in the thread of branches through memory looking like this:

A100 → 8192 → 7AFC → 717E → 7B31 → A108

Threading simply places the address of the current program step onto the dispatch stack. Then the a branch to the routine pointed to by this address is executed. When it has finished, the top address on the dispatch stack is popped, incremented by two bytes (to point to the next address in the current routine) and the process repeated.

Figure 5.

Recursion

Imagine that the procedures names A, B, and C exist and that a recursive procedure (one which calls itself) named Q has been written as follows:

```
PROC Q    A  B  C  Q    END
```

then when Q is invoked, the following sequence of events occur:

Address on dispatch stack	Address pointed to by dispatch stack	Depth of dispatch stack
Q+2	A	1
Q+4	B	1
Q+6	C	1
Q+8	Q	2
Q+2	A	2
Q+4	B	2
Q+6	C	2
Q+8	Q	3
...	...	...

Following each recursive call the thread lengthens. Only resolving the calls (i.e., going back up the thread) or using the EMPTY command will return the thread to the starting point.

Figure 6.

program once the code at a branch had completed execution.

APT uses a very efficient means of accomplishing this. It lays down a thread—a list of addresses prior to branching—onto an internal stack. Once the code branched to has been executed, APT can return to its previous position by popping this thread of addresses from the stack until it finds the first one, much like a child following a string to find the way home (Figure 5) and in a fashion reminiscent of MUMPS or Forth threading techniques. The module within APT which accomplishes this is called the dispatcher.

By carefully designing the dispatcher, this technique results in fast, resource-efficient programs. For example, using a command in a program compiles to only two bytes of code. Similarly, recursion is easily accomplished (Figure 6). The technique of threading, however, is used only within a procedure. Interprocedure links are handled quite differently.

All APT procedures are strung together as trees. These trees are created automatically by the compiler to link different procedures. As Figure 7 shows, this is done by reserving pointer space within each procedure. Pointer spaces contain the address of the next procedure in the tree to execute or of the next tree to execute. This is conceptually similar to the list structure in LISP.

Whenever the compiler is invoked, both procedures and the commands within them are resolved into threads and trees (Figure 8). Trees, as you can see, do not require the intermediate step of reference to a dispatcher to locate the next piece of code to execute. They are, therefore, even more rapid than threads. There is a cost however; extra memory overhead is required for pointer space.

APT overcomes this problem by reserving trees for inter- rather than intra-procedure communication. The extra memory required by pointer space therefore becomes negligible. (Knowledgeable readers will recognize in this scheme a combination of list processing, threading with pointers, and subroutine threading without the need for garbage-collection algorithms, excessive memory usage, or thrashing.) Tree creation is implicit. Trees are created automatically whenever a program consists of more than one procedure.

Explicit programmer control of trees is also possible, as we have seen. Hence the name APT—Automatic and Programmable Trees. An important point here is that all implicit tree structures (that is, created automatically by the compiler) can be explicitly manipulated by a program or programmer. This allows a compiled program to be altered at run time, producing a number of side benefits such as array equivalencing, program self-modification, and a number of other fea-

Trees—linking procedures

Intra-procedure statements require only two bytes of code (the address of the routine to be executed) since the dispatcher retrieves this address and executes a branch to it. Inter-procedure links between procedures, however, require four bytes (two for the branch and two for the address of the next procedure to branch to).

Program statement	Pointer	Space required (bytes)
PROC	dispatcher	2
2+3	dispatcher	6
?	dispatcher	2
END	direct jump to next proc	4

Regardless of their nature, all APT commands require two bytes of storage only. The exception to this is the END (and ENDASM) statement which requires four: two for a branch command, and two to act as a pointer to the next procedure. (If no procedure exists, the pointer bytes point to the interpreter routines.) For example:

```
PROC A "A"? END        PROC B "B"? END        PROC C "C"? END
PROC D "D"? END        PROC E "E"? END        PROC F "F"? END

PROC Q
     A B D F E C
END
```

When Q is created, the following structure is built:

Procedure	Pointer bytes contain address of
A	B
B	D
D	F
F	E
E	C

Figure 7.

Threads and trees in a program

Here are three simple programs:

```
PROC A 1+2 END     PROC B 2 END     PROC C 3 END
```

and here is a program to run all three:

```
PROC Q
     A C B
END
```

When Q is executed:

Program step	Address on dispatch stack (threaded call)	Pointer bytes (implicit tree)
Q		A (dispatcher not used)
1	number routine	
+	addition routine	
2	number routine	
END		C (dispatcher not used)
2	number routine	
END		B (dispatcher not used)
3	number routine	
END		Interpreter

Within a procedure, the dispatcher finds the address of the next routine to execute, saves the address containing this address on the dispatch stack, then jumps to that address. When the routine is finished the dispatcher pops the top address on its stack and uses it to find the address on the next routine, and so on (Figure 5). This creates threaded jumps under direction of the dispatcher.

Between procedures, however (e.g., when procedure A has completed executing), the address of the next procedure to execute is already stored in the body of procedure A. Therefore the dispatcher need not be called and a direct jump to procedure C is executed. This creates implicit trees.

Figure 8.

tures inherent in the language.

The third type of coding performed by the compiler is optimization. A general idea of what this means can be seen from the single example in Figure 9.

In this example an array-creation command is encountered in the input stream. The first thing the compiler or interpreter must do is reserve space for the array in memory. Since multidimensional arrays are usually stored sequentially in memory, a mapping function is used to locate the address of a requested cell relative to the first cell (Figure 9). This mapping function must be performed each time an array is referenced during execution of a program since there is no way to tell ahead of time what cell in the array will be requested. As Figure 9 illustrates, there are two methods for arriving at a cell of an array when the program references it—one involves using multiplication, the other vectoring. The first is memory efficient but slow, the latter memory inefficient but fast.

An optimization compiler has built within it the algorithms to resolve prob-

```
1. PROC A                    ; This procedure uses a stack for
                             ; argument passing.
2.   +5 DUP                  ; Add 5 to the current stack
                             ; contents, duplicate the result.
3.   IF <100 THEN A ENDIF    ; If the value on the stack is less
                             ; than 100, call A (recursively)
4. END                       ; End the procedure. Notice that no
                             ; variable storage has been used.

1. PROC B                    ; This procedure uses a variable to
                             ; pass arguments in its recursive
                             ; calls to itself.
2.   I+5->I                  ; Add 5 to I, store the result in I.
3.   IF I<100 THEN B ENDIF   ; If I<100 then call B again.
4. END                       ; End the procedure.
```

Listing 10.

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.

KCSystems
20 Lamington Drive, Succasunna, NJ 07876
(201) 927-9104

CIRCLE 45 ON READER SERVICE CARD

Only $95 with FULL SOURCE CODE!

Q/C

"... an incredible learning tool." *Byte*

For only $95, Q/C is a ready-to-use C compiler for CP/M with complete source code. Here's what *BYTE* (May 1984) said: "Q/C ... has a portable library and produces good code quality. If you want to learn compiler construction techniques or modify the standard language, Q/C is the obvious choice."

- Source code for compiler and over 75 library functions.
- Strong support for assembly language and ROMs.
- No license fees for object code.
- Z80 version takes advantage of Z80 instructions.
- Q/C is standard. Good portability to UNIX.

Q/C has casts, typedef, sizeof, structure initialization, and function typing. It is compatible with UNIX Version 7 C, but doesn't support long integers, float, parameterized #defines, or bit fields. Call about our new products: Q/C profiler, Z80 code optimizer, and Z80 assembler and virtual linker, all with full source code!

THE CODE WORKS
5266 Hollister, Suite 224
Santa Barbara, CA 93111
(805) 683-1585

Q/C, CP/M, Z80, and UNIX are trademarks of Quality Computer Systems, Digital Research, Zilog, Inc., and Bell Laboratories respectively.

CIRCLE 42 ON READER SERVICE CARD

lems such as this, that is, which one of the techniques should be compiled when accessing a memory cell. The APT compiler, for example, computes the space necessary to store vectors, then uses vectored array mapping as opposed to multiplicative mapping according to the ratio of this space to the total memory space available. Optimization in this context, then, refers to the ability of the compiler to choose an algorithm according to the context in which that algorithm will be required. Of course, the choosing happens at compile time, not at execution time.

APT is written entirely in APT, with two small exceptions: the I/O interface (roughly 500 bytes) and the controller (roughly 200 bytes).

Both of these are written in the assembler of the destination computer and combined will typically require less than 1K bytes of code.

APT purposefully avoids the Pascal p-code approach—using pseudo-assembler commands to interface to an actual assembler—because true optimization of the I/O and controller requires native mode, that is, machine-specific coding. The I/O interface consists of any special drivers required to map the I/O of the hardware onto the internal I/O handlers in APT. For example, if a computer contains floating point hardware, the I/O interface would map this hardware onto pointers within APT's floating point software.

The controller contains code to drive the dispatcher and to handle inter-procedure branching (trees). APT's speed is dependent upon the hardware stacks available and the addressing capabilities of the hardware. This is because the dispatcher makes use of up to three internal stacks and because both tree control and dispatching rely heavily upon directed branching throughout memory. It makes sense therefore to write the controller in the machine language of a particular computer to best use that computer's stack and addressing architecture.

Once these two small pieces of coding are written for a particular computer, the body of the language may simply be loaded onto the machine. This should make the implementation of APT on different types of computers quite straightforward.

There are three elements which are generally considered to be essential in a general purpose language. The language should contain a robust command set, be easy to use, and be efficient. As a result, APT contains many or all of the functions and capabilities found in FORTRAN, COBOL, Pascal, Forth, LISP, BASIC, and other readily available languages. It does this in an efficient, malleable, easy-to-use package.

APT's approach is to allow the programmer to work with whole structures rather than the typical one-thing-at-a-time of most other languages. Hierarchical and other data structures can be manipulated as wholes, broken apart, and reassembled into new structures at will. This helps make APT a cognitively rich language. Flexible and interactive, it adapts to the user, getting smarter as new procedures and dictionaries are added.

Yet it still retains its readability, ease of use, and efficiency (for example, it easily fits onto a 16K byte computer).

It was out of a desire for this kind of power in a language that I created APT. Version 0.1 of the language has been up and running for approximately three months on a seven-year-old Tandy Mod 1 computer. While only extended use will prove the system, few bugs have so far appeared. The only further additions that I plan to add to the language are improved string handling and relocatable object code. Both should be completed by the time you read this. Once I am satisfied that it is bug free, and I have completed an APT manual, it is my intention to release APT into the public domain.

Overall, APT has met all of its design goals nicely and has resulted in a general purpose language of some elegance and power. To find out more about APT, drop me a note c/o: *COMPUTER LANGUAGE*, 131 Townsend St., San Francisco, Calif. 94107, and I'll respond to your requests for information.

Optimization example

Multiplicative algorithm

Suppose the compiler encounters an array request:

INTEGER AA(100*5*20)

The array will be stored in a contiguous block of sequential memory of 100*5*20 WORDS (integers require two bytes, or one word, each) like so, for a total of 20K bytes:

cell number (each cell is two bytes long):

1	2	...	99	...	499	500	501	502	...	1000

To access the data in AA(1,2,3), an algorithm is used to translate the (1,2,3) into a number relative to the first cell in the array. This number is then added to the absolute memory address of that first cell to arrive at the desired data. The algorithm is:

(sub1 – 1) + (dim1 * (sub2 – 1) + (dim1 * dim2 * (sub3 – 1)))

where sub = subscript dim = maximum size of the array dimension

Vectoring algorithm

An alternative approach to calculating a relative cell address is to use an addition algorithm based on stored vectors. Suppose we have an array:

INTEGER AA(4,5)

cell number	1	2	...	5	6	...	9	10	...	20
subscript:	1,1	1,2	1,n	2,1	2,2	...	3,1	3,2	...	4,5

For this simple array, a vector N would be stored consisting of five elements:

n1	n2	n3	n4	n5
0	4	8	12	16

To locate the data in AA(a,b):

vector N(b) + a

For example AA(3,2) is the tenth cell relative to the start:

```
cell number  = N(3) + 2
             = 8    +2
             = 10
```

Figure 9.

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

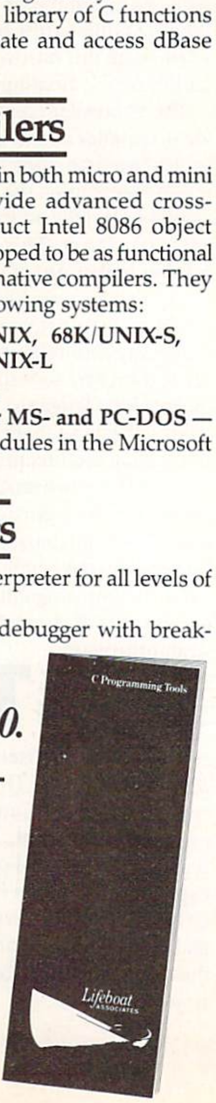
For programmers active in both micro and mini environments we provide advanced cross-compilers which product Intel 8086 object modules. All were developed to be as functional — and reliable — as the native compilers. They are available for the following systems:

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.

CIRCLE 85 ON READER SERVICE CARD

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

Gordon French—Old man of the micro revolution

By Regina Starr Ridley

"I guess I'm the only guy in the business who's been in it this long and hasn't made a million dollars," wryly chuckled Gordon French, a self-described aging hippie who has spent the past 10 years in the center of Silicon Valley's microcomputer revolution.

French's lack of a million dollars is one of the distinctions he may hold in the microcomputing industry. He also may have been involved in the largest number of microcomputer product and organization start-ups.

In early 1975 French, together with Fred Moore, founded the Homebrew Computer Club—the first club for microcomputer owners. While Homebrew was in its first year, French was closely involved in the development of one of the most exciting microcomputers at the time: Processor Technology's Sol. His next job was with another notorious start-up, IMSAI, where he opened and managed the first Computer Shack, which later became ComputerLand. Other job stints have included work at Exidy, Commodore, 3M, and CompuPro.

French is now about 50 years old and projects the impression of not having changed much over the years. His greyish-whitish hair grazes his collar, and he wears corduroys held up with suspenders. He has moved from a house in Menlo Park, Calif. with a machine shop and a huge garage to an equipment-filled, one bedroom apartment in nearby Sunnyvale. He worries about paying the rent.

Now he's considering writing a book about the early years of the microcomputer industry from his special perspective, which began when he became a computer operator in 1954.

Soon after graduating from high school, French went to work for White Stag, a sportswear manufacturer, in Portland, Ore. His first job was to calculate the pay of the seamstresses, who worked on a piecemeal basis. He spent his lunch times in the tabulating department and soon landed the job of keypunch operator—the only man among 17 women.

Soon after his big move to keypunch, White Stag installed a computer, which turned out to be the first Univac 120 in the Northwest, according to French. By 1954 he became a trainee TAB operator and has been around computers ever since.

Most of his work was in assembly language. "At one time in the early 1960s I found myself practically unemployable because I did not code in COBOL or FORTRAN," said French. But that kind of experience proved very helpful, he added, because when he went to work for Processor Technology, he and Steve Dompier were the only people who could sit down and write assembly language code that would do input and output.

French's jobs in the 1960s and early 1970s included work on special purpose apparatus at the AMES Research Center and a job with Planning Research Corp. certifying that RPG code on large projects was correct and operational.

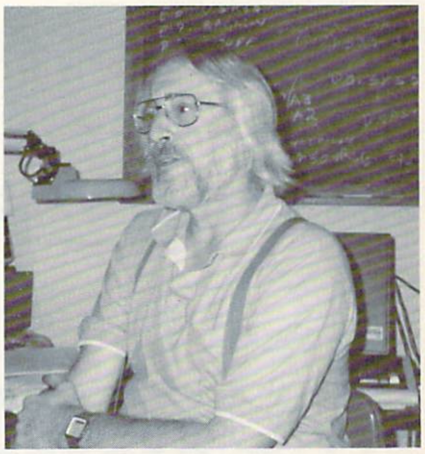
"I became probably the world's authority on RPG. That is so ludicrous! RPG is the kind of thing they put beginning people through to learn how to do simple reports. To make claim to be the world's authority on RPG is so ridiculous that it's absurd, but it was true," said French.

By the time 1975 rolled around, French was working for Microform Data Systems installing a huge system for the Social Security Administration in Maryland. He was working for them when he founded the Homebrew Computer Club.

The forerunner of Homebrew was Wednesday night potluck suppers organized by the People's Computer Company in Menlo Park. Due to some political turmoil on the board and the fact that "the women weren't very hot for doing the potluck," said French, "the thing was coming unstuck."

"Finally it was decided by the powers that be at the People's Computer Company not to hold the Wednesday potluck meetings any more," said French. "Fred Moore, who was trying to set up a network system with a data base of 3- by 5-cards in a shoe box, and I walked out on the sidewalk, and both of us felt that something had been lost."

"Fred asked where else we could meet, and I told him I had a big, warm garage in Menlo Park, why don't we meet there?



Fred, who's always short of money, said 'well, it's going to cost about $5 to get handbills printed.'

"I said fine and gave him the $5. He went off, did a little artwork for it, and posted it all over the place." The potluck suppers ended in November, and the first meeting of the new group was March 5, 1975."

"It rained to beat hell that day," remembered French, "but between 17 to 22 people showed up." Unfortunately, no accurate written record exists as to who attended, but French said the list included Bob Marsh, who was setting up Processor Technology; Lee Felsenstein, a consultant to Proc Tech; Steve Dompier, who was to write Target and Trek for the Sol; Bob Reiling, who was co-owner of the first West Coast Computer Faire and wrote the Homebrew newsletter; and Steve Wozniak, designer of the Apple computer.

The timing of Homebrew was impeccable. The first meeting was held just one month after Altair appeared on the cover of *Popular Electronics*. "Whammo!," said French, "here we were starting up a public club exactly at the time when you could buy for $400, or thereabouts, a machine!"

Moore brought an Altair to the second meeting. French described it as "a box with a cord coming out of a thing that you plugged in the wall, and when you turned the thing on the lights flashed." "But by the third meeting, Dompier found out that a transistor radio placed next to the CPU would begin to get radio frequency interference. He noticed that certain commands would give it certain tones and intonations."

This meeting was held at the Flood Mansion, a big house in Menlo Park built about 100 years ago. Electricity had been added around 1920, so there were few outlets. Plugging in the Altair required using a 150-ft extension cord running down the stairs and into the meeting room.

"Dompier spent the entire meeting handkeying a program on the Altair to play 'Fool on the Hill.' He entered all 256 bytes of this thing and was sitting there with the machine quietly humming, waiting for the meeting to end so he could play 'Fool on the Hill,' and some kid coming down the stairs kicked the cord out," groaned French, laughing.

"He re-entered it, and just by the time the meeting was over we were able to play 'Fool on the Hill.' There was much shouting, laughter, and applause. Everybody remembers that, and I believe that it's Wozniak's favorite story," said French.

French was conducting the meetings during that period, but in the summer of 1975 had to move to Woodlawn, Md., for three months to install the system at the Social Security Administration headquarters. Lee Felsenstein took over the chairmanship of Homebrew. About that time the meetings were moved to the Stanford Linear Accelerator Center auditorium or, if it were occupied, the cafeteria.

French was under pressure to move back East with his family, but "precipitously" quit his job on Friday, Dec. 14, 1975. By that Monday, he had a job with Proc Tech across the bay in Emeryville, Calif.

Proc Tech was started up by Bob Marsh and Gary Ingram to produce 4K Altair memory boards. Felsenstein, who wanted to remain independent, was a consultant.

"About the third or fourth day I was there," said French, "Felsenstein came in with a big drawing and pasted it up on the wall. In one corner of it were the letters SOL. SOL in the Northwest where I came from meant Shit Out of Luck, and I thought, what on earth are they doing with a name like that?" [The name is said to have been derived from the expression "having the wisdom of Solomon" and also was a reference to Les Solomon, technical editor of *Popular Electronics.*]

French described the Sol as the second integrated personal computer. "I am really sick and tired of the argument that keeps going on between Steve Jobs and I whether the Apple was the first. The Apple was the third. The first one was called the Sphere and was produced in Utah. Then came Sol followed by Apple."

At first the Sol was kept under tight wraps, said French. But eventually the project matured to the point where others had to become involved. French's major contribution was to build the sheet metal case or, as one author has put it, the "mechanical realization" for the original Sol.

"My hobby for years was building small-scale, live steam locomotives, and I was the only one around with any mechanical skills. In addition to my garage in Menlo Park, I had a large machine shop with several drill presses and lathes," said French.

French was also responsible for hooking up Proc Tech with Gene Tepper, the man who designed the look of the Sol. Dick Gray of Gray Associates drafted and created the sheet metal. "Gray is the kind of guy that designs gun mounts for fighter planes," said French. "Everything was re-enforced with double sheets and gussets for strength."

The sturdy case is the basis for one of French's favorite incidents. "At the office I never wore my shoes," he said. "One day I had my shoes off with the Sol down on the floor. I got up and stood on it. For some time, Marsh and Ingram talked about running an ad with me standing on the thing saying 'no Gordon, we meant stand behind it!'"

MEMO: *C Programmers*

QUIT WORKING SO HARD.

These people have quit working so hard: IBM, Honeywell, Control Data, GE, Lotus, Hospitals, Universities & Government Aerospace.

THE GREENLEAF FUNCTIONS™

THE library of C FUNCTIONS that probably has just what you need . . . **TODAY!**

. . . already has what you're working to re-invent
. . . already has over 200 functions for the IBM PC, XT, AT, and compatibles
. . . already complete . . . already tested . . . on the shelf
. . . already has demo programs and source code
. . . already compatible with all popular compilers
. . . already supports all memory models, DOS 1.1, 2.0, 2.1
. . . already optimized (parts in assembler) for speed and density
. . . already in use by thousands of customers worldwide
. . . already available from stock (your dealer probably has it)
. . . It's called the **GREENLEAF FUNCTIONS.**

Sorry you didn't know this sooner? Just order a copy and then take a break — we did the hard work. Already.

THE GREENLEAF FUNCTIONS GENERAL LIBRARY: Over 200 functions in C and assembler. Strength in DOS, video, string, printer, async, and system interface. All DOS 1 and 2 functions are in assembler for speed. All video capabilities of PC supported. All printer functions. 65 string functions. Extensive time and date. Directory searches. Polled mode async. (If you want interrupt driven, ask us about the **Greenleaf Comm Library**.) Function key support. Diagnostics. Rainbow Color Text series. Much, much more. **The Greenleaf Functions.** Simply the finest C library (and the most extensive). All ready for you. From Greenleaf Software.

. . . **Specify compiler** when ordering. Add $7.00 each for UPS second-day air. MasterCard, VISA, check, or P.O.

◆ Compilers:
CI C86 . . . . $349
Lattice $395
Mark Williams . . . $475

◆ General Libraries. . . . $175
(Lattice, Microsoft, Mark Williams, CI C86)
◆ DeSmet C $150
◆ Comm Library $160

GREENLEAF SOFTWARE©

GREENLEAF SOFTWARE, INC.
2101 HICKORY DRIVE ◆ CARROLLTON, TX 75006 ◆ (214) 446-8641

CIRCLE 44 ON READER SERVICE CARD

French asserted that one of the big inaccuracies in *Fire in the Valley*, Paul Freiberger and Michael Swaine's popular book on the development of the microcomputer, is that it credited the Sol exclusively to Marsh and Felsenstein.

"Felsenstein did the circuitry. Marsh's sole contribution was the keyboard, which was a delight. Outside of that, the rest was up to the staff. What I want is my due for the mechanical side of the Sol."

The Sol's big introduction to the world was at the Atlantic City, N.J., PC computer show in June 1976. "The night before we were to leave, we didn't even know how many of us were going," said French. "Money was quite tight. I went at the last minute because we were still installing the circuit board and getting the thing together. We got virtually no sleep the night before our 7 am flight."

"We had two Sols with us. One had all the electronics in it and was in a blue case. The other had been done as an appearance model and was in suntan tones or, as one of the hippies at Proc Tech described it, earth tones.

"The one in earth tones had no electronics in it at all. It was just a case with everything shoved up and poked in with cardboard and held in with tape. That one was put way in the back of the display once we got there."

French went out that night with a couple of people he met from Technical Design Laboratories and drank beer until around 3 am. No sooner had he quietly crawled into bed in the hotel room he was sharing with Felsenstein than he heard some banging.

"I thought, what the hell was that? Then I heard another bang and Felsenstein came in with the Sol under one arm and a scope under the other. What had happened was that we installed a tantalum capacitor in the power supply backward, and it finally gave out. Getting down to the power supply meant everything had to come apart.

"So it was back out of bed for me, and I was about three sheets to the wind. We took the damn Sol apart again to replace this capacitor and then put the thing back together again. I don't think Felsenstein and I got any sleep the second night."

It was worth it though because "we stole the show with the Sol," said French. The two big displays at the show were Proc Tech's and Micro Instrumentation Telemetry Systems' (MITS), manufacturer of the Altair.

French worked exactly one year—until Dec. 17, 1976—for Proc Tech. "I couldn't get a decent raise so I quit," he said. Immediately he began work for IMSAI, Proc Tech's number one competitor.

IMSAI was filled with adherents of Erhardt Sensitivity Training (est). "Bill Millard [IMSAI founder and president] broadly suggested that anybody who wanted to succeed at IMSAI would have to work until 10 pm, and I said, no way. My wife had taken a job that left me with two kids to put to bed every night. All the guys who had some sort of est training told me exactly the words to use: 'I choose not to do that.' I was the only one who went home at some kind of reasonable hour."

"Ed Faber, who was Millard's right-hand man, asked me what my career goals were. I told him I'd like to make some money and a few things like that. He told me he had two impossible jobs and would I like both of them. I thought he was joking. He wasn't!

"That day I got to be manager of product selection, evaluation, and test. The second job was pilot manager for the first Computer Shack, later to become ComputerLand."

Faber wanted French to "select some things that would really be flashy when the store opened." French also gathered some big names in the industry to attend the store opening, including Greg Yob, who wrote Hunt the Wumpus; Li-Chen Wang, author of Palo Alto Tiny BASIC; Ed Hall, a senior software designer; and Alice Algren, a Ph.D. with Cromenco, a new start-up.

French assembled these people in a small anteroom, while in the main room a

GOOD NEWS!

C for the 6809 WAS NEVER BETTER!

INTROL-C/6809,Version1.5

Introl's highly acclaimed 6809 C compilers and cross-compilers are now more powerful than ever!

We've incorporated a totally new 6809 Relocating Assembler, Linker and Loader. Initializer support has been added, leaving only bitfield-type structure members and doubles lacking from a 100% full K&R implementation. The Runtime Library has been expanded and the Library Manager is even more versatile and convenient to use. Best of all, compiled code is just as compact and fast-executing as ever - and even a bit more so! A compatible macro assembler, as well as source for the full Runtime Library, are available as extra-cost options.

Resident compilers are available under **Uniflex, Flex** and **OS9.**

Cross-compilers are available for **PDP-11/UNIX** and **IBM PC/PC DOS** hosts.

Trademarks:
Introl-C, Introl Corporation
Flex and Uniflex, Technical Systems Consultants
OS9, Microware Systems
PDP-11, Digital Equipment Corp.
UNIX, Bell Laboratories
IBM PC, International Business Machines

For further information, please call or write.

INTROL CORPORATION
647 W. Virginia St.
Milwaukee, WI 53204
(414) 276-2937

CIRCLE 32 ON READER SERVICE CARD

photographer was taking pictures of Millard, Faber, the mayor of Hayward and Miss Hayward. French asked the photographer to come into the next room to take "what probably would have been the most historic photograph in microprocessing history," but the photographer refused, saying he'd been chartered to take only seven photographs and already had taken eight.

The incident still aggravates French as if it were yesterday. "I should have gone to the drugstore and bought a camera!" he said with frustration. "That particular piece of foolishness and nonsense cost us a very historic document."

French left Computer Shack after a controversial change in management. He quickly went to work for Exidy, a another start-up. There he worked with Howell Ivy on the Sorcerer, a computer system which "was never a success." After the design was completed, he left.

That night happened to be a Homebrew meeting. It was the only night that Chuck Peddle, division manager of computer operations with Commodore and architect of the 6502, attended. "I stood up and asked if anyone wanted to employ an aging hippie," French said. "Not a soul stirred."

But after the meeting Peddle asked French to come to his office the next day. French became manager of customer applications software.

Management at Commodore was somewhat disorganized, and in a little less than a year French decided he'd like to run his own business. In 1977 he became a distributor for 3M. Pretty soon, says French, he was selling 2,000 diskettes a month out of his garage, about $120,000 worth of diskettes a year.

French urged 3M to go all out and rapidly expand its production of 5¼-in floppy disks. "They could have owned the market," said French, "because the only other manufacturer that could produce anywhere near the quality was Dysan, and Dysan has always been small production. 3M's shortsightedness in not understanding the exact nature of what I was telling them must have cost them, conservatively, $50 million."

While a 3M distributor, French did some consulting for the Japanese at $110 per hour. But he did not enjoy doing business with them, mostly because of cultural differences in conducting business.

French recalled early morning wake-up calls for urgent meetings at strange hours. Business was conducted with a strong emphasis on secrecy. He remembered the time a Japanese firm placed a 100,000-piece order through him with a U.S. company. The order was delivered by a semi into his driveway and precisely 20 min later a truck from an airfreight company owned by the Japanese company packed up the boxes for shipment straight to Japan.

"The money was good but I just don't want to do business with those guys," said French.

French sold the last of his 3M products in 1982 when 3M decided to get away from distributors and go directly to end users.

Since 1982 French says he's "pretty much done some sliding and gliding." He spent about five months working on the mechanical side of the CompuPro system 10. He currently works sporadically at a friend's Byte Shop in Hayward, Calif.

French also has kept busy working on the computer systems in his apartment, which include 21 microprocessors, 50 million bytes of storage capacity, six keyboards, six video display instruments, six printing devices, and 350 floppy disks.

He runs an active remote bulletin board system and recent conversations have focused on stories of the development of microcomputers. It helped spark an interest in French to write a book and set the record straight about that time of, as he described it, "extraordinary people doing extraordinary things."

The memories of that time are very important to French. A camaraderie was created that he still feels a part of and doesn't want to lose.

"Nobody forgets those days. No one who really was there," he said, almost wistfully. ■

Regina Starr Ridley is managing editor of COMPUTER LANGUAGE.

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

6 TIMES FASTER!

SuperFast Software Development Tools

INCREASE YOUR PROGRAMMING EFFICIENCY
with high-performance software development products from SLR Systems.
No other tools approach the speed or flexibility of the SLR Systems line.

"Z80ASM is an extraordinary product...",
Robert Blum, Sept. 84 DDJ

"...in two words, I'd say speed & flexibility",
Edward Joyce, Nov. 84 Microcomputing

ASSEMBLERS

- RMAC/M80 macros
- Nested INCLUDES & conditionals
- 16 char. labels on externals
- Built in cross-reference
- Optional case significance
- Phase/dephase
- Math on external words and bytes
- Define symbols from console
- Generate COM, HEX, SLR-REL, or Microsoft-REL files
- Time & Date in listing
- Over 30 configure options

	Z80ASM -full Zilog Z80	$125
NEW!	Z80ASM+ -all tables virtual	$195
NEW!	SLRMAC -full Intel 8080, with Z80.LIB extensions internal	$125
NEW!	SLRMAC+ -all tables virtual	$195

Z80 CPU, CP/M compatible, 32K TPA required.

"Z80ASM...a breath of fresh air...",
Computer Language, Feb. 85

LINKERS

- Links SLR & M80 format files
- Output HEX or COM file
- Three separate address spaces
- Load map and SID/ZSID .SYM file

SLRNK+ includes:

- All tables overflow to disk
- HEX files do not fill unused space
- Intermodule cross-reference
- EIGHT separate address spaces
- Works with FORTRAN & BASIC
- Generate PRL & SPR files
- Supports manual overlays
- Full 64K output

	SLRNK -fastest memory based	$125
NEW!	SLRNK+ -full featured virtual	$195

Combo Paks available from $199. - $299.

For additional information contact SLR Systems
1-800-833-3061, in PA (412) 282-0864
1622 N. Main St., Butler, PA 16001 • Telex 559215

MasterCard VISA C.O.D., Check or Money Order Accepted

SLR Systems

CIRCLE 73 ON READER SERVICE CARD

MODULA-2 PROGRAMMING TOOLS

A collection of utility modules ready to link into your programs and greatly speed programming efforts and the operation of programs.

Each tool is supplied as a definition module with in-line documentation, an implementation module with full source code and a ready-to-link object module. A fully-linked ready-to-run test program with source code is included.

Each module is implemented using Logitech's Modula-2/86™, Version 1.1 and MS-DOS/PC-DOS™ Version 2.0 or later unless otherwise specified. All modules are upward compatible with Microsoft's Xenix™ operating system as specified in the Microsoft MS-DOS Programmer's Reference Manual.

MemUtils: high-speed memory utilities coded using 8086 string instructions.
Keyboard: a complete IBM-PC keyboard handler.
ScreenOps: high-speed routines for controlling IBM-PC text screen. Based on ROM BIOS calls.
FileOps: direct access to MS-DOS file handling functions via DOS function calls.
DirOps: direct access to MS-DOS's hierarchial directories via DOS function calls.
DiskUtils: miscellaneous disk and drive utilities via MS-DOS function calls.
SingVD: calculates singular values of real-values matricies.
MicroMouse: direct access to all 16 Microsoft Mouse funtcions via mouse system software function calls.

Memutils	**$29**	All three for **$59**
Keyboard	**$39**	
ScreenOps	**$39**	
FileOps	**$39**	All three for **$79**
DirOps	**$39**	
DiskUtils	**$29**	
MicroMouse	**$49**	
SingVD	**$89**	

Developed by: Thomas H. Woteki, Ph.D.

Entire package of 8 modules - all with source code and test programs for **$189**

VISA MasterCard

Add $3/order shipping and handling   VA residents add 4% sales tax

Call **703/ 522-8898** or send your order to: **Information Systems Incorporated**
1901 No. Fort Myer Drive, Arlington, VA 22209

-Quality Software At Low Prices-   **-Save Time With Expert Tools-**

CIRCLE 20 ON READER SERVICE CARD

ORDER COMPUTER LANGUAGE BACK ISSUES WHILE THEY LAST!

Complete your collection of COMPUTER LANGUAGE magazines with our selection of back issues. A complete set is sure to become a valuable collector's item in years to come. Here are just a few of the features included in each issue:

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 1
- 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

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 $4.00 per issue.

Premier	______ copies × $4.00 = $ ______	NAME ______
Oct. '84	______ copies × $4.00 = $ ______	COMPANY ______
Nov. '84	______ copies × $4.00 = $ ______	ADDRESS ______
Dec. '84	______ copies × $4.00 = $ ______	CITY, STATE, ZIP ______
Jan. '85	______ copies × $4.00 = $ ______	
Feb. '85	______ copies × $4.00 = $ ______	
Mar. '85	______ copies × $4.00 = $ ______	
______	copies × $4.00 = $ ______	
	Total $______	

Send payment and coupon to: 4

COMPUTER LANGUAGE

Back Issues
131 Townsend St.
San Francisco, CA 94107

PRODUCT BINGO ⊗○⊗○⊗○⊗○

Each month Product Bingo features the latest in new software and hardware products of interest to COMPUTER LANGUAGE readers. Product Bingo items are based on information received from the manufacturer and are not meant to be product evaluations, reviews or endorsements. To find out more about a particular product simply circle the appropriate number on the Reader Service card—you'll receive information directly from the manufacturer.

Note to manufacturers: Send new product information to Doug Millison, Product Bingo, COMPUTER LANGUAGE, 131 Townsend St., San Francisco, Calif. 94107.

By Doug Millison

⊗○⊗○⊗

Intel development on the IBM PC

Software developers can now use the IBM PC to design software for Intel microprocessor-based systems. The **Universal Development Interface** from **RTSC** lets you run Intel 16-bit software on any MS-DOS-based computer system. With the communications software provided, Intel compilers and utilities can be uploaded or downloaded to and from an Intel Microprocessor Development System and an IBM PC or compatible. The developed software can then be downloaded from the PC to a target system.

In addition, RTSC offers the 86/88-based compilers and utilities, written for the Intel MDS ready to run under the Universal Development Interface, at half the price of the same software purchased directly from Intel.

Universal Development Interface requires a PC with 256K RAM and two 360K RAM floppies, or one floppy and a 10MB hard disk, and PC/MS-DOS version 2.0 or higher. Retail price is $500.

Address: RTSC, 1390 Flynn Rd., Unit E, Camarillo, Calif. 93010, (805) 987-9781.

CIRCLE 101 ON READER SERVICE CARD

⊗○⊗○⊗

Put an APL in your pocket

STSC Inc. offers **PocketAPL**, an inexpensive APL programming package for the IBM PC, PCjr, and true compatibles running PC-DOS and MS-DOS.

Priced at $95, PocketAPL comes with an introductory book, reference guide, keyboard reference and placard.

PocketAPL is upwardly compatible with **APL*PLUS/PC System**, an application development system from STSC. Containing programs for software development, screen management, report formatting, disk emulation, and communications, the APL*PLUS/PC System runs under MS-DOS or PC-DOS and sells for $295.

Address: STSC Inc., 2115 E. Jefferson St., Rockville, Md. 20852, (800) 592 0050, in Maryland (301) 984-5123.

CIRCLE 102 ON READER SERVICE CARD

⊗○⊗○⊗

You control the video

You'll add a dash of Gallic panache to your screen design with **Matis**, the screen manager from France via **Softway Inc.**

Matis allows you to design data entry/display screens of almost unlimited size; create windows, single or multiple screen display; and manage the screens while running the application. Matis commands let you control the video attributes of each object—lines, fields, text, windows, pages—independently.

For the IBM PC/XT and true compatibles equipped with DOS and 128K RAM, Matis is available under interpreted and compiled BASIC, Pascal, C, Assembler, and other popular languages. Suggested retail is $150.

Address: Softway Inc., 500 Sutter St., Ste. 222, San Francisco, Calif. 94102, (415) 397 4666.

CIRCLE 103 ON READER SERVICE CARD

⊗○⊗○⊗

UNIX-like programming tools

QTOOLS, a programmer's toolbox from **QCAD Systems Inc.**, is available for use under MS-DOS or PC-DOS on IBM PCs and compatibles.

Adapted from UNIX, the 19 QTools cover file listing, pattern search, substitution, translation, and file maintenance. QTools supports I/O redirection and pipes, wildcards, environment variables, command-line options, and on-line help.

QTools sells for $49.95.

Address: QCAD Systems Inc., 1164 Hyde Ave., San Jose, Calif. 95129, (408) 255-5574.

CIRCLE 104 ON READER SERVICE CARD

⊗○⊗○⊗

Clean up your language

Your Microsoft and Turbo Pascal programs will be easier to read, understand, and modify with **TIDY**, a Pascal formatter from **Major Software**.

Tidy transforms a raw input program into the recommended standard, highly structured form. With Tidy, formatting style can be ignored as the program is entered, speeding the program's creation. Tidy indents structures, capitalizes key words, formats declarations, sensibly places comments, and adds blank lines where they will best increase readability.

Tidy runs on IBM PC/XT/AT under PC-DOS or MS-DOS with 128K RAM. The Microsoft Pascal and Turbo Pascal versions retail for $69 and $49 respectively, plus $5 for shipping.

Address: Major Software, 66 Sylvian Way, Los Altos, Calif. 94022, (415) 941-1924.

CIRCLE 105 ON READER SERVICE CARD

Epsilon

The Emacs-Like Text Editor For Programmers Who Don't Like to Wait!!

State of the Art Text Editor

Epsilon is an exciting new text editor designed to make programmers more productive. Epsilon is faster than Brief, faster than Mince, faster than Gosling Emacs, *and faster than the editor you're using now!*

Concurrent Processes!

Epsilon lets you compile while you edit! You can run compilers, assemblers, linkers, and almost any other program that isn't screen oriented, all under Epsilon's control, while you edit your files!

With Epsilon you don't wait for programs like compilers to finish. Use Epsilon's concurrent process command, and while the compiler runs, you can continue to examine and edit files. Any errors in the compilation are displayed immediately, and Epsilon gives you the opportunity to correct them **while the compiler continues to run.** *With Epsilon, you're finished correcting errors when other editors first let you start.*

Powerful Commands

Epsilon has over 125 commands instantly available. Epsilon can manipulate words, sentences, and paragraphs easily. Epsilon will automatically save text you have deleted in a "ring" of kill-buffers, so that you can retrieve it later. It will help you avoid syntax errors by displaying matching parentheses. *And best of all, Epsilon's macros let you define your own commands, which can be loaded automatically each time you start Epsilon.*

Speed with No Limits.

Epsilon reads and writes files 25% to 600% faster than competing editors. From its convenient keyboard macros to its completion facility that completes the names of commands, files and buffers, to its optimized incremental search, Epsilon has been designed for programming ease and speed.

There's no limit to the number or the size of buffers you can have. Each buffer can hold a different file, or different versions of the same file. You can create as many windows as will fit on the screen, and display different buffers in each. And should you run out of memory, Epsilon will create and automatically utilize a swap file.

Speed Comparison with Other Editors

(Time in Seconds)

	Epsilon	Brief	Mince	Emacs
Start-up	2.60	4.11	1.43	24.93
Read 21K file	1.06	1.33	8.95	7.52
Write 21K file	2.11	14.30	6.05	7.95
Next Screen	.19	.24	1.33	1.80
String Search	3.85	7.04	4.49	8.41
I-Search	3.85	--	--	8.73
First Help	8.30	12.33	--	--
Other Helps	.20	11.64	--	--

Epsilon runs on IBM PC's, XT's, AT's and compatibles with PC-DOS 2.0 or above and requires 192K of memory.

Epsilon's price is only $195.00.

ALL MAJOR CREDIT CARDS ACCEPTED.

Lugaru Software, Ltd.
5227 Fifth Avenue, Suite 12 / P. O. Box 110037
Pittsburgh, Pa. 15232

(412) 621-5911

CIRCLE 51 ON READER SERVICE CARD

Write it once!

MasterFORTH

Portable programming environment

Whether you program on the **Macintosh**, the **IBM PC**, an **Apple II** series, a **CP/M** system, or the **Commodore 64**, your program will run unchanged on all the rest. If you write for yourself, MasterFORTH will protect your investment. If you write for others, it will expand your marketplace.

MasterFORTH is a state-of-the-art implementation of the Forth computer language. Forth is interactive – you have immediate feedback as you program, every step of the way. Forth is fast, too, and you can use its built-in macro assembler to make it even faster. MasterFORTH's relocatable utilities, transient definitions, and headerless code let you pack a lot more program into your memory. The resident debugger lets you decompile, breakpoint, and trace your way through most programming problems. A string package, file interface, and full screen editor are all standard features.

MasterFORTH exactly matches the Forth-83 Standard dialect described in *Mastering Forth* by Anderson and Tracy (Brady, 1984). The standard package includes the book and over 100 pages of supplementary documentation.

MasterFORTH standard package	
Macintosh	$125
IBM PC and PC Jr. (MS DOS 2.1)	125
Apple II, II+, IIe, IIc (DOS 3.3)	100
CP/M 2.X (in several formats)	100
Commodore 64	100
Extensions	
Floating Point (1984 FVG standard)	$40
Graphics (Apple II series)	40
Module relocator (with utility sources)	60
Printed source listing (each)	35
Publications	
Mastering Forth (additional copies)	$18
Thinking Forth by Leo Brodie	16
Forth-83 International Standard	15
Rochester Bibliography, 2nd ed.	15
1984 Rochester Conference	25
1984 J1 of Forth Appl. & Res. 2(2)	15
1983 FORML Conference	25

MICROMOTION

12077 Wilshire Blvd., #506
Los Angeles, CA 90025
(213) 821-4340

CIRCLE 56 ON READER SERVICE CARD

Microcomputer COBOL compiler analysis

By Charles K. Ballinger

COBOL is an old-timer computer language. With its roots dating back to the early 1960s, few doubt whether it is a survivor.

Up until approximately 1978 COBOL's range extended through the mini to mainframe environment due to the size of the compilers and their tremendous appetite for both disk space and memory. Now that micros are available with 512K to over 1MB of memory, you will probably see all computer languages grow in size as the designers no longer are bound by memory restrictions.

Additionally, this change makes it possible to find COBOL on microcomputers. The current microcomputer COBOL compilers are now capable of providing almost all of the features found in the mainframe environment. Several of the compilers are now rated by the General Service Administration (GSA) at the high-intermediate level based on the 1974 ANSI COBOL standard.

Many of you may be wondering why anyone would implement a mainframe language on a micro when C, Pascal, and others are available and most run much faster. However, COBOL was designed for business and since the bulk of microcomputers are being used in a business environment, it is the logical language choice.

Let's begin

This wrap-up of the currently available COBOL compilers and interpreters for both MS-DOS and CP/M will focus not so much on the speed of the compilers as on the features they offer the software developer or in-house programmer. We will cover 12 compilers for PC-DOS/MS-DOS and one for the CP/M environment. Basic information on the compilers is contained in Table 1.

Fortunately, or unfortunately, depending on your viewpoint, MS-DOS seems to be gaining momentum and is rapidly replacing CP/M as the operating system of the future. Although I only received one compiler for the CP/M environment, Microsoft, Ryan-McFarland, Micro Focus and Digital Research all either sell or distribute compilers for the CP/M environment.

This wrap-up is not a detailed review of these compilers but instead is a general overview of the strengths and weaknesses of each compiler. Each compiler will have its own section with references made to the variety of tables I compiled during the course of the review. Following that will be a brief overview of the various royalty charges currently required and a summary of this article.

While many of the versions I received for review were beta or prerelease versions, I'm not going to dwell on bugs or errors I discovered during the course of my benchmarks. This is done for two reasons. First of all, since each vendor has beta test sites, I will assume that the errors have been reported and are now in the process of being corrected. Secondly, it is not fair to give you, the reader, a false impression of problems when I am not reviewing the same software you would receive at order time.

The benchmarks used in this review are listed in Table 2 with descriptions of the tests each performs. If you'd like a copy of the actual benchmark you can obtain it from either the *COMPUTER LANGUAGE* Bulletin Board Service or from the magazine's account on CompuServe (type "GO CLM").

Perhaps in other reviews you have read speed has been a critical consideration. This is due in part to the fact that both C and Pascal are system-oriented rather than business-oriented languages. When it comes to system functions or utilities, speed is foremost on your list of requirements.

Writing in COBOL is a different matter. Remember, COBOL is left with the day-to-day drudgery processing. How creative can you be when writing an accounts receivable or payroll system in COBOL? COBOL is the workhorse of business while C and Pascal can share the limelight for graphics, games, special utilities, and the like.

Since COBOL programs usually access fairly large data files and are used to kill many trees with printout, speed becomes of secondary importance.

COBOL was originally a batch-oriented language. This means the data coming into the computer was normally entered via punchcard and not via terminal. With users demanding faster turnaround on their information, it is now essential to go to an interactive mode.

With this new concept implemented in COBOL, the operator becomes the slowest link in the chain. With interactive screen handling the program must now wait for the operator to complete data entry before the program can process the data. This places the speed of the compilers in a different light. Now, ease of data manipulation via the terminal replaces speed as the most important feature of a COBOL compiler.

Personally, as a software developer I look for ease of use, ability to easily convert existing software to the micro environment, and manufacturer support as a few of the most important features. One additional feature I also look for is the ability to port the software I've written to a variety of systems while making few, if any, changes.

The compilers presented in this review have been tested on an IBM PC-XT with 640K of RAM running PC-DOS version 2.0. While several of the vendors offer generic MS-DOS compilers, I received only PC-DOS versions so could not test some of the others for their particular implementation under generic MS-DOS.

I tested the CP/M compiler on my Z100 running CP/M 85 with 704K of RAM installed and 10MB of hard disk and running at 4.77MB clock rate. In all cases the timings I obtained are approximates and should not be held to the second.

On the IBM I have a clock card installed so the timings should be fairly accurate. For the timings on the compile and link steps, as well as the run times for the CP/M tests, I had to rely on a stopwatch. So in your assessment do not omit the related human error factor that such a device introduces. In those compilers requiring the link step to be a separate function, I timed them as a separate entry so the compile times may be misleading unless you take the link-step time into account.

It is my hope that by the time you have finished this review, you can decide for yourself which compiler would suit your particular task and proceed from there. The tables provided will give you an overview of the speed comparisons, the options supported, and other factors that should assist you in your decision on a compiler.

Digital Research level II COBOL v. 3.0

This product is a fusion of something from both Digital Research Inc. and Micro Focus. The basic portions of the compiler are identical to the Micro Focus level II compiler with the exception that DRI has added an access manager and display manager to its version of the compiler. In all the benchmark programs shown in Table 4, you will note that both Micro Focus level II and DRI level II produced identical .GNT modules. Execution speed was identical with the exception of program PCPERF, which ran 1 sec slower under the DRI level II version. Compile

Bundled software and essential information

Manufacturer	Name	Price	Version	Operating system	Linker	Debugger	Cross reference	SORT option	Copy protected	Editor	Network support	Royalty fee	Compiler output	Pages in manual
Digital Research	Level II	$ 700	3.0	PC-DOS	no	no	no	incl.	no	no	no	no	.INT, .GNT	575+
Ellis Computing	Nevada COBOL	$ 29.95	2.1	CP/M 80	N.A.	no	no	no	no	no	no	no	.OBJ	165
mbp Software & Systems Technology	mbp COBOL	$ 750	7.40	PC-DOS	no[2]	yes	yes	incl.	no	no	no	yes	.OBJ	740+
mbp Software & Systems Technology	mbp COBOL	$1,000	9.00	PC-DOS	no[2]	yes	yes	incl.	no	no	yes[5]	yes	.OBJ	753+
Micro Focus	Professional COBOL	$3,000	1.0 not avail.	PC-DOS	yes	yes	yes	incl.	no	built-in	yes[5]	yes	.GNT, .INT	594+
Micro Focus	Professional COBOL	$3,000	1.1	PC-DOS	yes	yes	yes	incl.	no	built-in	yes[5]	yes	.INT, .GNT	600+
Micro Focus	Level II	$1,500	2.6.2 2.5	PC-DOS, MS-DOS, CP/M, UNIX	yes, builder	no[3]	yes	incl.	no	no	no	yes	.INT, .GNT	569+
Micro Focus	VS-COBOL	$4,000	1.0	PC-DOS, MS-DOS	no	yes	yes	incl.	no	built-in	yes[5]	yes	.INT, .GNT, .COM	600+
Microsoft	Microsoft COBOL	$ 700	1.12	MS-DOS	yes	yes	no	$200	no	no	no	no	.EXE	600+
Microsoft	Microsoft COBOL	$ 700	2.0	MS-DOS, UNIX 286*	N.A.	yes	no	incl.	no	no	yes	no	.INT	650
Realia	Realia-COBOL	$ 995	1.0 1.2 in Apr.	PC-DOS 2.0+	no[2]	yes	yes	no	yes	yes	no	no	.EXE	198+
Ryan-McFarland	RM-COBOL	$ 950	2.0B	PC-DOS MS-DOS	no	yes	no	$100–$300	no	no	yes	yes	.COB, .OBJ,	476+
WATCOM	WATCOM COBOL	$1,500[1] $3,000[1] $ 250 one-time charge	2.1	PC-DOS	N.A.	yes	no	no	no	yes	no	N.A.	N.A.	712

*UNIX version to be released soon.

N.A.—Not Applicable or Not Supported.

1. Only available to educational or commercial institutions since the compiler is licensed on a yearly basis. Covers unlimited copies for the licensed site location.
2. Uses linker as supplied on the IBM DOS diskette. 3. Option purchase item animator package is $1,200, forms package is $300.
5. To run under a network may require the purchase of the networked version or additional support modules not normally supplied with the compiler.

Table 1.

time was somewhat slower than the native Micro Focus version and it should be noted that while Micro Focus level II allows you to output either .INT or .GNT files, the DRI version automatically generates only .GNT files as compiler output.

The DRI version supports basically the same features that the native Micro Focus version did except in the areas of screen handling and file I/O. Under the DRI method you now can use the popular access manager and display manager to handle screen formatting and both random and indexed sequential access method (ISAM) file access.

Changing from Micro Focus level II to DRI level II could be time-consuming if your application does considerable screen formatting or file I/O. If, however, you are migrating applications from the CP/M environment that were written using either the access manager or display manager, your conversion should be quite easy.

This product should be DRI's mainstay in the COBOL department since the company is now locking buyers into its product line. Without the access manager or display manager, applications developed in DRI level II are not automatically transportable to Professional COBOL of Micro Focus.

Ellis Computing Nevada COBOL v. 2.1

Ellis Computing's compiler was the only one I received for testing under CP/M, although many of the vendors listed in this wrap-up supply a compiler for the CP/M environment. The Nevada COBOL compiler produces .OBJ code which must then be run in much the same fashion as the new version of Microsoft COBOL and RM-COBOL.

Only seven of the benchmark programs would compile due in part to lack of support for various verbs used within the COBOL programs. The compiler does not support the *COMPUTE* verb. Although it can be remedied with the use of the basic commands of *ADD*, *SUBTRACT*, *DIVIDE*, etc., it would change the timings of the benchmarks. The failure of several of the table programs was the compiler's inability to handle the *VARYING* or *INDEXED* verb within this particular implementation.

These benchmarks were tested on my Z100 under CP/M 85 so were running on the 8085 CPU at a 4.77MB clock rate. Even at this rate this compiler came up dead last in the timing of execution speed. I'm not sure how much CP/M affected this situation, but it is hard to make a comparison against the other compilers when you consider the cost difference.

Overall this compiler is an excellent choice if you are still running CP/M on your system—it will run under an 8080,

Benchmarks used

Program name	Description
PCPERF	Execute 10,000 *PERFORM* statements as a null routine
PCGOTO	Execute 10,000 *GOTO* statements
PCADDSUB	Execute 10,000 *ADD* and *SUBTRACT* statements
PCINTADD	*ADD* integers 32,676 times
PCMLTDIV	Do 1,000 *MULTIPLY* and *DIVIDE* operations
PCMOVE	Execute 10,000 *MOVE* statements
PCIF	Execute 10,000 *IF* statements
PCSTRING	Execute 10,000 string concatenations
PCILOOK	Read a 100 element table 100 times using *INDEXING*
PCSLOCK	Read a 100 element table 100 times using *SUBSCRIPTING*
SIEVE	Sieve of Eratosthenes prime number routine as discussed in *BYTE*, January 1983, p. 283. Does two interations of the routine for timings.
PCFIBNCI	Performs 24 Fibonacci-number generation interations 100 times
PCMATRIX[1]	Simulated program of Matrix benchmark—which tests floating point arithmetic speed—for C compilers (see *COMPUTER LANGUAGE*, February 1985, p. 82). Issue of subscripts must start at 1 for COBOL and value routines were modified accordingly.

1. Of the benchmarks that have now appeared across languages, the PCMATRIX program is the only one that can only approximate the functionality of the same benchmark in either C or Pascal. Please do not make any attempt at comparison between languages on this particular program since it can be heavily tainted depending on how you wish to implement the code.

Table 2.

Program Editing Breakthrough

The first reconfigurable editor to combine windows with undo and a macro language — BRIEF

Paul Callahan, information systems manager for a major Boston bank, says that "BRIEF is the only editor that I am aware of that incorporates all of the features preferred by most programmers:"

■ Full UNDO (N Times)	■ Windows (Tiled and "Pop Up")
■ Edit Multiple Large Files	■ Unlimited File Size
■ True Automatic Indent for C	■ Reconfigurable Keyboard
■ Exit to DOS Inside BRIEF	■ Online Help
■ Uses All Available Memory	■ Search for "regular expressions"
■ Intuitive Commands	■ Mnemonic Key Assignments
■ Tutorial	■ Horizontal Scrolling
■ Repeat Keystroke Sequences	■ Comprehensive Error Recovery
■ 15 Minute Learning Time	■ Compiled MACRO Language

Availability: PCDOS-compatible systems, AT, Tandy.

Try BRIEF. Use the Demo...or the full product for 30 days. Only $195. Call **800-821-2492.**

Solution Systems

BRIEF is a trademark of UnderWare.

335-Z Washington St., Norwell, MA 02061

CIRCLE 94 ON READER SERVICE CARD

8085, or Z80—and you are interested in the possibilities that COBOL offers. Even if you are just learning the language, this compiler is a good choice. Just be aware that it is somewhat limited, so don't condem COBOL based on Nevada COBOL's lack of features.

The manual that accompanies the compiler is handy and provides adequate information for the new programmer to start using the compiler. Several sample programs are also provided to assist you in your learning.

This compiler does not support the *SORT* verb and to do debugging you must make use of the "D". feature in column 7 to put out appropriate debug messages. While very simplistic, it is priced right for the individual interested in learning COBOL without spending a fortune.

mbp COBOL v. 7.40

mbp's compiler version 7.40 was the fastest compiler on the market as of two months ago and would have retained that honor had Realia COBOL not entered the picture. This early version of the compiler was amazing. For the size of the compiler and the amount of disk space it ate up, it produced some of the fastest programs I have ever seen on a micro.

When I first reviewed this product several months ago, I had to run some of the benchmarks twice just to be sure they were actually executing. The strengths this version had have been passed on to its successor. In addition, the amount of disk space it uses remains the same.

In order to run this compiler in a development mode, you must be running in a hard disk environment. Although you can run it using floppies, I have yet to subject myself or anyone else to a 20 min compile while changing five diskettes. In order to accomplish any serious development,

Compile time comparisons (sec)

COBOL compiler	PCPERF	PCGOTO	PCADDSUB	PCINTADD	PCMLTDIV	PCMOVE	PCIF	PCSTRING	PCILOOK	PCSLOOK	SIEVE	PCFIBNCI	PCMATRIX
DRI level II v. 3.0[3]	80	82	81	79	80	86	82	96	90	87	119	92	113
Ellis Computing Nevada v. 2.1	10	10	10	10	11	11	11	N.C.	N.C.	N.C.	N.C.	N.C.	N.C.
mbp v. 7.4[1]	119/20	115/20	115/20	110/20	113/20	120/22	115/20	114/20	114/20	120/20	114/20	115/20	138/20
mbp v. 9.0[1]	74/12	74/12	76/12	78/12	88/12	83/12	83/13	74/13	79/13	78/13	87/14	80/12	80/12
Micro Focus Professional v. 1.0[2]	39	41	42	40	41	44	43	48	46	43	56	45	70
Micro Focus Professional v. 1.1[2]	20	22	23	23	26	28	29	35	29	31	57	35	34
Micro Focus level II v. 2.6.2[6]	14	14	14	13	14	16	15	15	16	16	17	16	14
Micro Focus VS-COBOL v. 1.0	20	20	20	20	20	20	20	20	20	20	20	20	20
Microsoft v. 1.12[1,4]	23/20	20/20	20/20	20/20	22/20	22/20	22/20	20/20	23/22	23/22	27/23	22/20	35/24
Microsoft v. 2.0[3,4]	24	25	25	26	31	30	31	31	31	31	37	31	41
Realia v. 1.0[1,5]	14/14	15/14	14/14	15/14	15/14	16/14	16/14	16/14	15/14	15/14	16/14	14/14	17/14
Ryan-MacFarland RM-COBOL v. 2.0B	20	20	26	27	28	36	28	N.C.	36	35	35	34	23
WATCOM v. 2.1	N.C.	N.C.	N.C.	N.C.	N.C.	N.C.	N.C.	N.C.	N.C.	N.C.	N.C.	N.C.	N.C.

N.C.—Would not compile or function not supported by compiler.

1. Requires separate link step using the IBM DOS linker. 2. Required that the code first be checked prior to compile. 3. Link or code generation is an integral part of the compile step. 4. Uses MS-LINK supplied with the compiler. 5. Compile time reported by the compiler, link time approximate. 6. Produces either .INT or .COM files as output, which may be executed.

All timings, with the exception of the Realia compile time, were hand timed using a stopwatch. Please take this into consideration when comparing timings that are less than 2 sec apart.

Table 3.

SQUEEZE MORE OUT OF EVERY ON-LINE MINUTE.

WITH NEW VIDTEX™ COMMUNICATIONS SOFTWARE FROM COMPUSERVE.

Presenting the software package that makes your computer more productive and cost-efficient.

CompuServe's new Vidtex™ is compatible with many personal computers sold today (including Apple,® Commodore,® and Tandy/Radio Shack® brands). And it offers the following features*–and more–to let you communicate more economically with most time-sharing services (including CompuServe's Information Service).

Auto-Logon. Lets you log on to a host simply and quickly by utilizing prompts and responses defined by you. Also allows quick transmission of predefined responses to host application programs after logging on.

Function Keys. Let you consolidate long commands into single keystrokes. Definitions can be saved to and loaded from disk file, allowing multiple definitions for multiple applications.

Error-Free Uploading and Downloading. CompuServe "B" Protocol contained in Vidtex lets you transfer from your computer to CompuServe and from CompuServe to your computer anywhere in the country. Also provides error-free downloading from CompuServe's extensive software libraries.

Full Printer Support. Printer buffer automatically buffers characters until printer can process; automatically stops on-line transmission when full; and automatically resumes transmission when capacity is re-established. Also, lets you print contents of textual video screen or RAM buffer at any time.

Capture Buffer. Saves selected parts of a session. Contents can be written to a disk file; displayed both on and off line; loaded from disk; and transmitted to the host.

On-line Graphics. Integral graphics protocol displays stock charts, weather maps and more.

If you are already a CompuServe subscriber, you can order Vidtex on line by using the GO ORDER command. Otherwise, check with your nearest computer dealer; or to order direct, call or write:

CompuServe

P.O. Box 20212, 5000 Arlington Centre Blvd.
Columbus, Ohio 43220

1-800-848-8199
In Ohio, call 614-457-0802

An H&R Block Company

CIRCLE 7 ON READER SERVICE CARD

*Some versions of the Vidtex software do not implement all features listed.

you'd better have at least 5MB of free space on your hard disk before you start your project.

This compiler does produce excellent output. The cross-reference feature, while not used each and every time, can be of great benefit when searching a long listing for a data name or a procedure.

The feature I found most annoying was the fact that this compiler does not conform to the standard COBOL *SORT/MERGE* convention and lacks *INPUT/OUTPUT PROCEDURE* features within the sort phase. Since its implementation of the sort is done via a call to a sort program, the user is very limited in the standard usage of the *SORT* verb as used in a mainframe COBOL. Conversion of the sort to this compiler's method is easily accomplished, but it still proves to be an annoyance.

Overall, this compiler is an excellent choice for the software developer who needs fast execution and a relatively easy method of screen design. mbp has implemented a screen management system (SMS) that reminds me of my CICS days. In mbp's system you generate screens as a separate entry and then use the *CALL* function to do screen I/O. Using this approach, you can keep your screen code external to your actual program code.

This version executed all the benchmarks with no problems. When you compare the size of the .EXE modules produced (Table 4), you see that they are relatively large in comparison to Microsoft's or Realia's. This compiler was the slowest of all tested in the actual compile phase. This is probably due to the sheer size of the compiler and the number of overlays necessary. Actual space required is somewhere between 1MB to 1.5MB of disk space, depending on file sizes and work-file requirements.

Comparison of benchmark programs on the IBM PC XT running PC DOS 2.0 (sec)

COBOL compiler	PCPERF	PCGOTO	PCADDSUB	PCINTADD	PCMLTDIV	PCMOVE	PCIF	PCSTRING	PCILOOK	PCSLOOK	SIEVE	PCFIBNCI	PCMATRIX
DRI level II v. 3.0	10	10	11	31	9	4	4	24	11	11	61	5	41
Ellis Computing Nevada v. 2.1	52	54	102	361	18	11	23	N.C.	N.C.	N.C.	N.C.	N.C.	N.C.
mbp v. 7.40	1	1	19	4	15	2	3	7	1	19	153	4	57
mbp v. 9.0	1	1	19	4	15	2	3	8	1	20	161	4	60
Micro Focus Professional v. 1.0	10	10	11	31	9	4	2	24	11	11	61	5	41
Micro Focus Professional v. 1.1	9	9	11	29	9	4	2	20	10	10	58	5	40
Micro Focus level II v.2.6.2													
.GNT	9	10	11	31	9	4	2	24	11	11	61	5	41
.INT	17	23	44	60	13	15	12	163	41	43	304	16	96
Micro Focus VS-COBOL v. 1.0													
.INT	14	18	42	52	12	15	10	28	19	41	296	16	94
.COM	14	18	42	51	12	15	10	28	19	41	296	16	94
Microsoft v. 1.12	20	40	73	223	19	11	17	88	113	93	756	22	243
Microsoft v. 2.00	32	64	93	366	28	36	39	131	240	191	1,374	50	454
Realia v. 1.0	1	1	2	5	3	1	1	2	1	2	21	1	19
Ryan-MacFarland RM-COBOL v. 2.0B	9	11	50	123	21	11	15	N.C.	72	66	465	17	135
WATCOM v. 2.1	N.C.	N.C.	N.C.	N.C.	N.C.	N.C.	N.C.	N.C.	N.C.	N.C.	N.C.	N.C.	N.C.

N.C.—Would not compile or function not supported by compiler.
All timings reported by programs with exception of CP/M version. These were hand timed using a stopwatch since CP/M does not support a date/time function in Nevada COBOL.

Table 4.

mbp COBOL v. 9.0

Version 9.0 is the latest one from mbp. As you will note in Table 1, the price has been increased from version 7.40 by $250. With the increase in price come many enhancements, including user-configurable defaults, command-line parameter passing, and a progress report. As you will see in Table 5, the size of the modules when compared to the prior version have been reduced by about 1K to 2K per module. This is due to the fact that a system interface module (SIM) now stays resident in RAM and helps reduce module size as well as the time required to load and execute a program.

The benchmark programs all compiled and executed on this version as well. The compile phase has been reduced by about 20 sec per module when you compare the compile time against version 7.4 (Table 3). Link time was also reduced but not as drastically, about 8 sec per program.

Unlike Microsoft, mbp's latest version maintained run times in all but four modules. The Sieve program takes 8 sec longer than it used to and the program PCMATRIX only ran 3 sec longer.

This version is a notable improvement over version 7.4 since many of the omissions found in version 7.4 have been corrected. Although the *SORT/MERGE* verb is still not directly supported, mbp has added an *OPEN SUBROUTINE* option, address and key-out options, as well as ISAM sort capability to its callable *SORT* routine.

You also have the option to use either the standard linker supplied with DOS or the PLINK linker available from Phoenix Software, which allows overlay management. This version also supports the ability to *CHAIN* to system commands (*COPY ERASE* etc.), and the *CHAINR* function allows you to chain to another program and have that program remain resident since you can now have more than 64K of data in memory at one time.

This version also supports the Novell Netware OS ($500 additional) for those interested in a networking environment.

Micro Focus Professional COBOL v. 1.0

Professional COBOL version 1.0 is certified at the highest possible level by the GSA. Even the communications section is supported, although not usable unless you are in a networked environment.

This compiler did extremely well in speed comparisons (Table 4) and was about average in the compile phase. Note in Table 5, you have the option to produce either .GNT or .COM files. The .COM files are stand-alone programs and have the necessary run-time system (RTS) support included in the module. Micro Focus is changing the usage of run time to application support module (ASM) since if you call *SORT* or ISAM, you still access the run-time module for these functions.

This compiler had no problem with any of the benchmarks, however, it should be noted that you must use the *CHECK* feature prior to compiling a program. If you attempt to compile a program without checking it first, the compiler will report that it cannot find the module.

As the price would indicate, this compiler has all the features you can think of and is entirely menu-driven via function keys. When coupled with the ability to design your screens using the forms package and then checking program flow with the annimator, it is no wonder a programmer could be very productive using this package.

The ease with which an application can be developed using this product became apparent shortly after I started using it. Although the documentation is very complete, I had to make little reference to it since everything was accessible via the screen. Help menus are even included should you have a question at any point along the way. You simply press the help function key, and you'll get a description that applies to the spot where you are currently.

My compliments to Micro Focus on an excellent product, even if it is slightly expensive for the average software developer.

Micro Focus Professional COBOL v. 1.1

DRI markets this compiler under the Micro Focus name but has no direct ties with it. Professional COBOL 1.1 does not currently support DRI's access or dis-

C + C UTILITY LIBRARY = PRODUCT

• We have over *200 complete, tested, and, documented functions*. All source code and demo programs are included.

• The library was specifically designed for software development on the IBM PC, XT, AT and compatibles. There are *no royalties*.

• Over 95% of the source code is written in C. *Experienced programmers* can easily "customize" functions. *Novices* can learn from the thorough comments.

We already have the functions you are about to write

Concentrate on software development—not writing functions.

THE C UTILITY LIBRARY includes:
• Best Screen Handling Available • Windows • Full Set of Color Graphics Functions • Better String Handling Than Basic • DOS Directory and File Management • Execute Programs, DOS Commands and Batch Files • Complete Keyboard Control • Extensive Time/Date Processing • Polled ASYNC Communications • General DOS/BIOS gate • And More •

• The Library is compatible with: Lattice, Microsoft, Computer Innovations, Mark Williams and DeSmet.

C Compilers: Lattice C—$349, Computer Innovations C86—$329; Mark Williams C—$449.

C UTILITY LIBRARY $149

Order direct or through your dealer. *Specify compiler* when ordering. Add $4.00 shipping for UPS ground, $7.00 for UPS 2-day service. NJ residents add 6% sales tax. Master Card, Visa, check or P.O.

ESI
ESSENTIAL SOFTWARE, INC
P.O. Box 1003 Maplewood, New Jersey 07040 914 762-6605

CIRCLE 18 ON READER SERVICE CARD

play manager software.

Like the version 1.0 from Micro Focus, this compiler had no problems with any of the benchmarks and although the execute times for the benchmark programs are identical, it did compile the programs faster than version 1.0 did. In fact, the compile time was reduced by about 50% when compared to version 1.0. Note in Table 5 that the .GNT modules stayed the same but the .COM output is approximately 2K to 3K larger.

This compiler produces everything you could ask for. Additionally, it does all this using very little disk space.

All comments applying to the Micro Focus version 1.0 also hold true here. Although I received a prerelease version of this package, I must assume that Micro Focus will be releasing this in its finished form shortly, possibly as you are reading this review.

Micro Focus level II v. 2.6.2

Level II COBOL version 2.6.2 experienced no problems with any of the benchmark programs and, in fact, compiled as fast as the Realia compiler. In this version you also have the ability to stop after producing an .INT file, which may be run, or continuing on and producing a .GNT file. The .GNT version, as you can see from Table 4, produces the fastest code of the two, but the .GNT modules are approximately twice the physical size of the .INT modules.

With the exception of the PCPERF program, the execution times were identical to the Professional COBOL times. PCPERF was 1 sec faster in level II than when run under the Professional COBOL compiler.

Level II can be raised up to the professional level by purchasing the forms package and the annimator package. However, by the time you have done that, you have spent as much as if you had just purchased the Professional version to start with.

This compiler is good for those who can't afford the professional version now but may someday migrate up to it. All code is upwardly transportable and this is handy when it comes time to move to a bigger system. With the logical progression between the versions that Micro Focus has implemented, it is easy to see why this series of compilers rates at the top.

Execution module size comparison (bytes)

Benchmark name	PCPERF	PCGOTO	PCADDSUB	PCINTADD	PCMLTDIV	PCMOVE	PCIF	PCSTRING	PCILOOK	PCSLOOK	SIEVE	PCFIBNCI	PCMATRIX
DRI level II v. 3.0 .GNT	3,712	3,840	3,968	3,712	3,840	4,096	3,840	4,352	4,224	4,096	12,672	4,096	11,264
Ellis Computing Nevada CP/M	2,048	2,048	2,048	2,048	2,048	2,048	2,048	N.C.	N.C.	N.C.	N.C.	N.C.	N.C.
mbp 7.40 .EXE	12,642	12,642	12,690	12,642	12,754	13,506	13,634	13,170	12,946	14,162	22,706	14,018	21,634
mbp v. 9.0 .EXE	11,038	11,036	11,076	11,032	11,128	11,334	11,157	11,200	11,348	11,410	19,952	11,413	18,729
Micro Focus Professional v. 1.0 .GNT	3,456	3,584	3,712	3,456	3,840	4,096	3,968	4,352	4,224	4,096	12,672	4,096	11,264
.COM	51,792	51,936	52,408	51,792	52,176	52,432	52,304	52,688	52,560	52,432	61,008	52,432	59,600
Micro Focus Professional v. 1.1 .GNT	3,456	3,584	3,712	3,465	3,840	4,096	3,968	4,352	4,224	4,096	12,672	4,096	11,264
.COM	55,152	53,520	53,632	53,376	53,760	54,016	53,888	54,272	54,144	54,016	62,592	54,016	61,104
Micro Focus level II v. 2.6.2 .INT	1,536	1,536	1,536	1,536	1,536	1,536	1,536	1,792	1,536	1,536	9,984	1,792	8,192
.GNT	3,712	3,840	3,968	3,712	3,840	4,096	3,968	4,352	4,224	4,096	12,672	4,096	11,264
Micro Focus VS-COBOL v. 1.0 .INT	2,048	2,048	2,048	2,048	2,048	2,048	2,048	2,048	2,408	2,408	2,304	2,048	2,560
.COM	51,888	51,888	51,888	51,888	51,904	52,128	52,032	52,016	52,096	52,096	60,352	52,048	58,416
Microsoft COBOL v. 1.12 .EXE	8,320	8,320	8,320	8,320	8,320	8,064	8,448	7,936	8,576	8,576	16,896	8,576	15,360
Microsoft COBOL v. 2.0 .INT	1,514	1,526	1,536	1,532	1,570	1,742	1,612	1,676	1,818	1,808	10,110	1,742	8,478
Realia v. 1.0 .EXE	6,241	6,239	6,273	6,523	6,843	6,521	6,389	6,389	6,475	6,657	14,981	6,487	14,135
Ryan-MacFarland RM-COBOL v. 2.0B .COB	1,024	1,024	1,280	1,280	1,280	1,536	1,280	N.C.	1,280	1,280	1,280	1,280	1,792
WATCOM v. 2.1	N.C.	N.C.	N.C.	N.C.	N.C.	N.C.	N.C.	N.C.	N.C.	N.C.	N.C.	N.C.	N.C.

N.C.—Would not compile, or function not supported by compiler.
Where more than one run-time module is available, they are both shown. The larger of the two would normally run in a stand-alone environment, as it incorporates the necessary run-time linkage and support.

Table 5.

TOTAL CONTROL:

FORTH: FOR Z-80®, 8086, 68000, and IBM® PC

Complies with the New 83-Standard

GRAPHICS • GAMES • COMMUNICATIONS • ROBOTICS
DATA ACQUISITION • PROCESS CONTROL

- ***FORTH*** programs are instantly portable across the four most popular microprocessors.
- ***FORTH*** is interactive and conversational, but 20 times faster than BASIC.
- ***FORTH*** programs are highly structured, modular, easy to maintain.
- ***FORTH*** affords direct control over all interrupts, memory locations, and i/o ports.
- ***FORTH*** allows full access to DOS files and functions.
- ***FORTH*** application programs can be compiled into turnkey COM files and distributed with no license fee.
- ***FORTH*** Cross Compilers are available for ROM'ed or disk based applications on most microprocessors.

Trademarks: IBM, International Business Machines Corp.; CP/M, Digital Research Inc.; PC/Forth+ and PC/GEN, Laboratory Microsystems, Inc.

FORTH Application Development Systems include interpreter/compiler with virtual memory management and multi-tasking, assembler, full screen editor, decompiler, utilities and 200 page manual. Standard random access files used for screen storage, extensions provided for access to all operating system functions.

Z-80 FORTH for CP/M® 2.2 or MP/M II, $100.00; **8080 FORTH** for CP/M 2.2 or MP/M II, $100.00; **8086 FORTH** for CP/M-86 or MS-DOS, $100.00; **PC/FORTH** for PC-DOS, CP/M-86, or CCPM, $100.00; **68000 FORTH** for CP/M-68K, $250.00.

FORTH + Systems are 32 bit implementations that allow creation of programs as large as 1 megabyte. The entire memory address space of the 68000 or 8086/88 is supported directly.

PC FORTH +	$250.00
8086 FORTH + for CP/M-86 or MS-DOS	$250.00
68000 FORTH + for CP/M-68K	$400.00

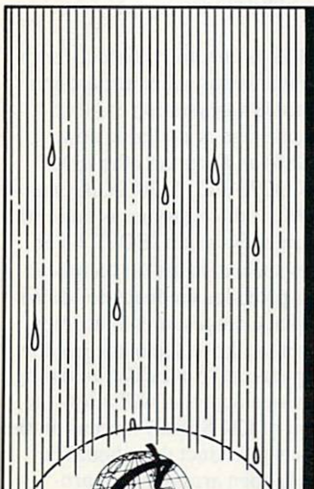
Extension Packages available include: software floating point, cross compilers, INTEL 8087 support, AMD 9511 support, advanced color graphics, custom character sets, symbolic debugger, telecommunications, cross reference utility, B-tree file manager. *Write for brochure.*

VISA

Laboratory Microsystems Incorporated
Post Office Box 10430, Marina del Rey, CA 90295
Phone credit card orders to ***(213) 306-7412***

MasterCard

CIRCLE 35 ON READER SERVICE CARD



An Optimizing, hassle-free C Compiler for the 8086-8088.

Ecosoft's Eco-C, the performance leader among full C compilers for the Z80, is now available for the 8086-8088 running under MSDOS (2.0 or later). Eco-C has features not found in any other 8086-8088 C compiler.

★ Over 100 library functions. Since they follow UNIX standards, your programs are highly portable in "both" directions ("up" to a UNIX machine or "down" to our Z80 compiler). This means new markets for your software at minimum development cost.

★ A single library. No more "dual" libraries and trying to remember what has to be linked with what. Your programs automatically take advantage of an 8087 if one is present at runtime.

★ A single floating point answer. No more "fuzzy floating point": your programs produce the same answers whether the floating point is done in hardware (8087) or software.

★ Error messages in English. No more cryptic numbers to look up. We tell you where the error occurred, what was found there, and what should have been there.

★ Strict syntax parsing. LINT is going to uncover fewer surprises because our parser looks hard at the details.

★ Efficient code. The optimizer pass of the compiler generates assembler code in Intel mnemonics that rivals that produced by compilers costing twice as much.

The price of the Eco-C compiler is $250.00 (all versions), including the user's manual, and is designed for use with Microsoft's MASM (or compatible) assembler and linker. When ordering, please specify disk format and whether you want the Z80-CP/M or 8088-MSDOS version of Eco-C.

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

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 Visa Mastercard
Add $10. shipping and handling

HARVARD SOFTWORKS

PO BOX 2579
SPRINGFIELD, OH 45501
(513) 390-2087

CIRCLE 47 ON READER SERVICE CARD

Micro Focus VS-COBOL
This compiler is the latest one out the gate from Micro Focus. It is so new that I only received a preliminary release version for testing, so please take this into consideration.

You will immediately notice that the compile time for VS-COBOL is somewhat misleading. Since the compiler I received did not support the actual *COMPILE* phase, I had to first *CHECK* the program and then use the *BUILD* facility in order to generate the actual .COM version of the program. The actual timings for this compiler ran from 9 to 15 sec for the *CHECK* function and approximately 8 sec to build the .COM version using the *BUILD* function provided.

You will see from Table 5 that the timings fall somewhere between the Professional version and level II COBOL when it comes to .INT module size generation. When compared with the .COM version the Professional compiler produces, there is only an average of 800 bytes difference between module sizes. What was most surprising was that except for one benchmark, all the timings were identical for the .INT and .COM versions.

Here again the benchmarks are fairly deceptive. This compiler is not intended for the software developer. It is instead an ideal compiler for large mainframe shops that either incorporate PCs in their data processing department or plan to do so.

With this compiler you can relieve the burden on the mainframe while increasing programmer productivity. Since this compiler supports full IBM VS-COBOL syntax, you have a very effective tool with which you can develop mainframe applications without tying up expensive mainframe resources. You have the ability to stress test your code without waiting for mainframe time since you can store a complete session of keystrokes that can be played back at a later point in time.

When used with an IBM PC-3270, IBM XT, or IBM AT connected to the mainframe, you have the ability to download code from the mainframe, develop the code on the micro, test the code on the micro, then upload the final program for actual production testing. VS-COBOL currently supports simulation of *CALL*s to CICS, IMS, DL/1, SQL/DS, and DB2.

By the time this appears in print, Micro Focus should have released an update to this package that will offer emulation of these services. The *COMPILE* feature will also be added shortly. Again, this is not intended to develop micro software. Instead it is meant as a mainframe tool to increase programmer productivity. This compiler provides syntax checking for OS/VS COBOL based on the 1974 standard as well as VS-COBOL II, IBM's new COBOL compiler incorporating elements of the yet-to-be-finalized ANSI 8X standard.

VS-COBOL is currently the only compiler on the market intended for use in a truly distributed workstation environment. This package is available now, (January 1985) and supports the IBM PC, XT, AT, 3270-PC, 3270-PC/G, and 3270-PC/GX. If the customer is under the Early Customer Program (ECP), he or she receives special status for technical support and provides feedback to Micro Focus on what changes or enhancements he or she would like to see in the package.

Current plans call for a CICS mapping support enhancement that will allow the development of CICS BMS screens on the micro. This is a product that should be received with open arms by data processing managers as an effective tool to increase productivity within their department at a reasonable price.

For programmers not wanting to learn a new editor, the product will even be integrated with the micro/SPF mainframe-like editor so programmers will not have to cross-learn a different editor. Learning curve time on this product is extremely short, which is another factor management must take into consideration.

Microsoft COBOL v. 1.12
This version of the Microsoft MS-DOS compiler corrected many of the bugs and minor errors found in the previous version

LATTICE WORKS

GSS SELECTS LATTICE FOR GRAPHICS SOFTWARE

Graphic Software Systems (GSS) and Lattice, Inc. recently joined forces to offer VDI & GKS graphics, based on the emerging ANSI standard, for the C language. As a result, Lattice is marketing graphics-based software that takes full advantage of the capabilities of personal computers.

According to Lattice President David A. Schmitt, "this coupling will allow the direct application of creative ideas without the constraints of many large-scale systems. Software development firms can develop graphics programs faster and at less cost."

The two companies also agreed to a continued sharing of technology which will support the development of standardized graphics software and ease the distribution of products.

LATTICE UNVEILS FOUR PRODUCTS

Lattice has announced the availability of four new software products for MS-DOS environments:

C-SPRITE is a software tool that simplifies debugging of programs written in Lattice C or assembly language. Cost: $175 per copy.

LMK is an Automated Product Generation Utility (UNIX "MAKE") that enhances productivity and relieves the tedium of rebuilding complex software systems or documents. Cost: $195 per copy.

The TEXT MANAGEMENT UTILITY PACKAGE includes utilities to search a set of files for simple or complicated patterns, to see the exact minimal differences between two text files, and to modify one or more text files automatically. Cost: $120 per copy.

CVUE is a full screen text editor that supports all normal screen editor functions and includes a configuration program to define tabstop positions, horizontal scrolling and edit commands. Cost: $100 per copy.

For complete information on these new products, contact Lattice.

LATTICE C NAMED 'BEST OF 1984'

The Lattice C compiler has been rated 'Best of 1984' by PC Magazine. According to columnist Peter Norton, "The Lattice C compiler is quite good . . . and in my opinion noticeably better than any of its competitors. Lattice C generates code that is quite compact and fast running; the closest competitor in my tests generated code that was about 10 to 15 percent bulkier."

ASK ABOUT OUR "TRADE UP TO LATTICE C POLICY"

After purchase, return registration cards for free subscription to the "Lattice Works" newsletter and important information about the Lattice Users Group

Lattice, Inc.
P. O. Box 3072
Glen Ellyn, IL 60138
(312) 858-7950
TWX 910-291-2190

CIRCLE 48 ON READER SERVICE CARD

(version 1.07). The most noticeable change was the correct handling of the *occurs* clause, which enables programs such as the Sieve benchmark to run, something Microsoft's CP/M version still does not allow.

This compiler ran all of the benchmarks with no problems. Unfortunately, if you are into speed, Microsoft COBOL version 1.12 is at the end of the line (Table 4). While this may seem like a drawback initially, it is not the case. In the 30,000-plus lines of code I have written with this compiler, I have found that the compiler is very easy to use, screens are a snap to produce or change, and the ISAM access speed is more than adequate for all business applications I have written to date.

The final output of the compiler is an .EXE file that requires the run-time module but produces relatively small code files. The major strengths of the Microsoft product is the ability to configure it for a variety of systems other than the IBM and that the run-time, screen configure programs, and utilities can all be distributed royalty free.

The compiler supports the standard COBOL *SORT/MERGE* verb, and you can do tag sorts as well as full-record sorts. The ability to handle the *SORT* statement in the same fashion as IBM mainframe COBOL makes this an easy version to which to convert mainframe code. You will notice in Table 5 that the .EXE modules are comparatively small and run well from a floppy or hard disk. It is important to remember that even though you may be developing software on your hard disk, not everyone is as fortunate and many still only have two disk drives.

This product's weakness is its inability to have multiple keys or indexes for a file. The other seeming omission in this version of the compiler, when compared to the company's CP/M version, is the inability to access any of the MS-DOS functions such as *DELETE* or *RENAME*.

The documentation is relatively good as documentation goes but fell far short when it came to helping me develop the 8088 code necessary to handle the *DELETE* and *RENAME* functions.

In the price vs. performance field, Microsoft COBOL version 1.12 is an excellent compiler and, as you can see by the newest version, Microsoft continues to enhance its product line.

Microsoft COBOL v. 2.0

This compiler represents a major deviation from the prior Microsoft COBOL versions. No longer producing .EXE files, the compiler instead produces .INT files in much the same manner as RM-COBOL does. So you must now enter *RUNCOB* <program name> in order to execute a program. This compiler now has validation at the highest possible level. The only items it does not support are the communications module and the report-writer module.

Like Microsoft's version 1.12, this version had no trouble with any of the benchmarks. As you compare execution speeds as shown in Table 4, you will note one disconcerting fact. This version produces even slower code than version 1.12 did. It did, however, reduce the .INT size of the modules by an average of over 7K per module (Table 5) when compared to the .EXE sizes. I would suspect that this causes somewhat more overhead within the run-time modules, which would account for the increase in run times.

In some cases this version took two to three times longer to execute. This version now supports multikey ISAM, split keys, and duplicate keys, all important features when being used in a business environment. The ISAM routine is now RAM-resident and must be loaded prior to being used. While this does take about 30K from the system, the speed of ISAM is noticeable.

Although not a part of this review, I did convert some of my current ISAM applications just to see what the differences were. Here again, the speed of the benchmarks is misleading. While the benchmarks would indicate this compiler is slower than a turtle, I found quite the

The C Interpreter:
Instant-C™

Programming in C has never been Faster.
Learning C will never be Easier.

Instant-C™ is an optimizing interpreter for the C language that can make programming in C three or more times faster than using old-fashioned compilers and loaders. The interpreter environment makes C as easy to use and learn as Basic. Yet ***Instant-C***™ is 20 to 50 times faster than interpreted Basic. This new interactive development environment gives you:

Instant Editing. The full-screen editor is built into ***Instant-C***™ for immediate use. You don't wait for a separate editor program to start up.

Instant Error Correction. You can check syntax in the editor. Each error message is displayed on the screen with the cursor set to the trouble spot, ready for your correction. Errors are reported clearly, by the editor, and only one at a time.

Instant Execution. ***Instant-C***™ uses no assembler or loader. You can execute your program as soon as you finish editing.

Instant Testing. You can immediately execute any C statement or function, set variables, or evaluate expressions. Your results are displayed automatically.

Instant Symbolic Debugging. Watch execution by single statement stepping. Debugging features are built-in; you don't need to recompile or reload using special options.

Instant Loading. Directly generates .EXE or .CMD files at your request to create stand-alone versions of your programs.

Instant Floating Point. Uses 8087* co-processor if present.

Instant Compatibility. Follows K & R standards. Comprehensive standard library provided, with source code.

Instant Satisfaction. Get more done, faster, with better results. ***Instant-C***™ is available now, and works under PC-DOS, MS-DOS*, and CP/M-86*. Money back guarantee within 30 days.

Find out how ***Instant-C***™ is changing the way that programming is done. ***Instant-C***™ is $495. Call or write for more information.

Rational
Systems, Inc.

(617) 653-6194
P.O. Box 480
Natick, Mass. 01760

Trademarks: MS-DOS (Microsoft Corp.), 8087 (Intel Corp.), CP/M-86 (Digital Research, Inc.), Instant-C (Rational Systems, Inc.)

CIRCLE 72 ON READER SERVICE CARD

GROWING OLD?

...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.
- Compiler Compatible--You can access functions and externals compiled with 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 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)*

CIRCLE 38 ON READER SERVICE CARD

opposite to be true when used in an actual production situation. To use this new version you must be running MS-DOS/PC-DOS version 2.xx or higher since portions of the run time remain resident. You need at least a 192K system and dual floppies.

Most important at installation time for this compiler is that you must have a CONFIG.SYS file on your system that has the *FILES = 10* statement in it. After I configured the compiler using the INSTALL program I could not figure out why the compiler would get to the code generation step and then fail under the pretext of "OVERLAY NOT FOUND." The *files* statement is absolutely necessary before the compiler will successfully generate code.

Microsoft COBOL version 2.0 compiled and generated .INT code for all of the benchmark programs with no problems. In spite of the poor showing this compiler puts up when compared to the others, I still favor this compiler for the ease with which you can design and implement screens as well as for the fact that no run-time royalties are required on distributed code.

Added features, such as its easy installation on a variety of terminals and systems and the ability to specify alternate keys, make this a compiler well worth the money. The price remains at $700 even with the sort feature now incorporated into the package. This version of the compiler supports the IBM network, and support for other networks are in the works. I must admit that I am somewhat prejudiced by this compiler since I have generated over 50,000 lines of code using version 1.12 in the last 12 months and found it very easy to use and maintain.

Realia COBOL v. 1.0

The Realia compiler has now edged out mbp COBOL as the fastest COBOL compiler on the market. While this compiler is not currently certified by the GSA (as of late January 1985), it will be by late March 1985. As you read this review, Realia's compiler should have received certification, and testing should be completed for its support of the Novell network.

Realia's COBOL version 1.0 is intended for the programmer who is very familiar with the inner workings of COBOL, which the company's manual reflects. It is a reference manual only and not intended to walk you through anything. Since Realia's compiler is intended to support the same coding structure as IBM VM-COBOL, you may need to acquire the manuals directly from an IBM publication center.

Realia's compiler was the only one tested that supported the time clock while the compiler was running. Each pass of the compiler is reported as to the time it started, and a full listing of all options selected are directed to the output file. This compiler produces one of the most extensive outputs I have seen.

It truly appears as if the output were produced on a IBM 370/158. This compiler has a multitude of options that can be set either via the command line or as the first line of the COBOL source program. While most options may not be used every day, they do represent almost every possibility you could ask for.

The strong points of this compiler are its amazing speed for sheer computations and the ability to produce native code in the form of a stand-alone .EXE file. The compiler also gives you full access to all DOS functions via a DOS interface. With this interface you can access any of the standard DOS functions, such as get date/time, or change date/time, make a directory, change a directory, etc. While this type of coding would tend to lock you into a very IBM compatible system you could still generate code for some MS-DOS machines.

In the limited testing I did, I found that this compiler had to run on an extremely IBM compatible system. When I attempted to run the output on my Z100 under MS-DOS 2.11 I ended up with a runaway machine displaying "WILD INTERRUPT". Since the Z100 is only about 75% compatible and not compatible at all in the area of screen I/O, I must assume that the target system must appear very much like an IBM PC in order to have the code execute correctly.

While this compiler is very fast, it only has rudimentary screen I/O capability. You, the programmer, are left with the task of handling any and all screen cursor positioning. Although this can be done, it would put a definite crimp in any development speed.

If full-screen I/O is not something you normally have to contend with, this compiler definitely should be considered. It has a full-screen editor included as part of the compiler, so you can generate source code without the requirement of another editor.

This compiler does not currently support any type of sort feature, so if a sort is a must then you must turn to another compiler. At the point in time this compiler implements full level II support it will become a well-rounded compiler. The only major areas that I think would need consideration are the *SORT/MERGE* feature and some easy way to handle full-screen I/O in order to reduce development time.

Realia's manual may well explain the compiler's implementation of screen handling, but I was left confused when I tried to follow the explanation provided. As you can see from Tables 3, 4 and 5, this

ConIX™

UNIX™ Technology for CP/M™

ConIX can provide any 48K+ CP/M-80 compatible system with many advanced capabilities of UNIX. You'll be amazed at what your 8-bit micro can do now! ConIX features include:

I/O Redirection and Pipes (uses memory or disk), multiple commands per line, full upper/lower case and argument processing, Auto Screen Paging, Programmable Function Keys, improved User Area Directory manipulation, Command and Extension (Overlay) Path Searching, "Virtual" disk system, 8Mb Print Spooler, extensive preprocessed "Shell" command programming language, 300+ variables, over 100 built-in commands, Math Package, 22 new BDOS SysCalls, Archiver (compacts files for disk space savings of over 50%), On-Line Manual System, and much more! Uses as little as 1/2K RAM! Runs with CP/M for true data and software compatibility. Installs quickly and easily without any system modifications.

The ConIX Operating System
List Price: $165

Price includes Instructional Manual, 8" SSSD disk, and free support. 5¼" format conversions available. To order, contact your local dealer, or buy direct and add shipping: $4.50 UPS, $10 Canada, $25 overseas. COD $2 extra (USA only). NY State residents add sales tax.

Computer Helper Industries Inc.
P.O. Box 680 Parkchester Station, NY 10462
Tel. (212) 652-1786

Dealer inquiries invited!

UNIX: AT&T Bell Labs, CP/M: Digital Research, ConIX: Computer Helper Ind.

CIRCLE 12 ON READER SERVICE CARD

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 INQUIRIES 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

NEW FOR THE MACINTOSH

MACINTOSH IS A REGISTERED TRADEMARK OF APPLE COMPUTER INC.

CIRCLE 13 ON READER SERVICE CARD

RP/M T.M.

By the author of Hayden's "CP/M Revealed."

New resident console processor RCP and new resident disk operating system RDOS replace CCP and BDOS without TPA size change.

User 0 files common to all users; user number visible in system prompt; file first extent size and user assignment displayed by DIR; cross-drive command file search; paged TYPE display with selectable page size. SUBMIT runs on any drive with multiple command files conditionally invoked by CALL. Automatic disk flaw processing isolates unuseable sectors. For high capacity disk systems RDOS can provide instantaneous directory access and delete redundant nondismountable disk logins. RPMPIP utility copies files, optionally prompts for confirmation during copy-all, compares files, archives large files to multiple floppy disks. RPMGEN and GETRPM self-install RP/M on any computer currently running CP/M®2.2. Source program assembly listings of RCP and RDOS appear in the RP/M user's manual.

RP/M manual with RPMGEN.COM and GETRPM.COM plus our RPMPIP.COM and other RP/M utilities on 8" SSSD $75. Shipping $5 ($10 nonUS). MC, VISA.

118 SW First St. - Box G
Warrenton, OR. 97146

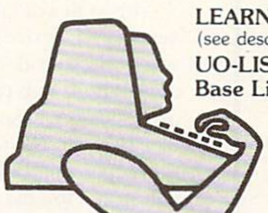
Micro Methods, Inc.

(503) 861-1765

CIRCLE 52 ON READER SERVICE CARD

(LISP) FOR A.I.

UO-LISP Programming Environment
The Powerful Implementation of LISP
for MICRO COMPUTERS



LEARN LISP System (LLS.1) (see description below	*$39.95*
UO-LISP Programming Environment Base Line System (BLS.1)	*$49.95*

Includes: Interpreter, Compiler, Structure Editor, Extended Numbers, Trace, Pretty Print, various Utilities, and Manual with Usage Examples. **(BLS.1)** expands to support full system and products described below.

UO-LISP Programming Environment: The Usual LISP Interpreter Functions, Data Types and Extensions, Structure & Screen Editors, Compiler, Optimizer, LISP & Assembly Code Intermixing, Compiled Code Library Loader, I/O Support, Macros, Debug Tools, Sort & Merge, On-Line Help, Other Utility Packages, Hardware and Operating System Access, Session Freeze and Restart, Manual with Examples expands to over 350 pages. Other UO-LISP products include: LISPTEX text formatter, LITTLE META translator writing system, RLISP high level language, NLARGE algebra system. Prices vary with configurations beyond (BLS.1) please *send for FREE catalog.*

LEARN LISP System (LLS.1): Complete with LISP Tutorial Guide, Editor Tutorial Guide, System Manual with Examples, Full LISP Interpreter, On-Line Help and other Utilities. LEARN LISP fundamentals and programming techniques rapidly and effectively. This system does not permit expansion to include the compiler and other products listed above.

LISP Tutorial Support (LTS.1): Includes LISP and Structure Editor Tutorial Guides, On-line Help, and History Loop. This option adds a valuable learning tool to the UO-LISP Programming Environment (BLS.1). *Order (LTS.1) for* **$19.95.**

REQUIRES: UO-LISP Products run on most Z80 computers with CP/M, TRSDOS or TRSDOS compatible operating systems. The 8086 version available soon.

TO ORDER: Send Name, Address, Phone No., Computer Type, Disk Format Type, Package Price, 6.5% Tax (CA residents only), Ship & Handle fee of $3.00 inside U.S. & CN, $10 outside U.S., Check, Money Order, VISA and MasterCard accepted. With Credit Card include exp. date. Other configurations and products are ordered thru our *FREE catalog.*

Northwest Computer Algorithms
P.O. Box 90995, Long Beach, CA 90809 (213) 426-1893

CIRCLE 46 ON READER SERVICE CARD

C

Software Development

PCDOS/MSDOS

Complete C Compiler
- Full C per K&R
- Inline 8087 or Assembler Floating Point, Auto Select of 8087
- Full 1Mb Addressing for Code or Data
- Transcendental Functions
- ROMable Code
- Register Variables
- Supports Inline Assembler Code

MSDOS 1.1/2.0 Library Support
- All functions from K&R
- All DOS 2.0 Functions
- Auto Select of 1.1 or 2.0
- Program Chaining Using Exec
- Environment Available to Main

c-window™ Symbolic Debugger
- Source Code Display
- Variable Display & Alteration Using C Expressions
- Automatic Commands
- Multiple Breakpoints by Function & Line Number

8088/8086 Assembler
- FAST — Up to 4 times Faster than IBM Assembler
- Standard Intel Mnemonics
- Compatible with MSDOS Linker
- Supports Full Memory Model

8088 Software Development Package **$199.00**

Includes: C Compiler/Library, c-window, and Assembler, plus Source Code for c-systems Print Utility

c-systems

P.O. Box 3253
Fullerton, CA 92634
714-637-5362

CIRCLE 16 ON READER SERVICE CARD

compiler had the shortest compile time under MS-DOS, the fastest execution times, and extremely small .EXE modules for run time.

The major complaint I have about this compiler is that even in a hard disk environment you must place the floppy disk in drive A to use the compiler (copy protected). I find this annoying and wonder what happens if the disk is damaged and you must wait for a replacement disk until you can continue to compile. You could be down for several weeks while waiting for the diskette.

Aside from this aspect, Realia's compiler is the one to watch in the coming months as new developments are underway.

Ryan-McFarland COBOL v. 2.0B

This compiler managed to run all of the benchmarks with the exception of PCSTRING. It seems that the *STRING* and *UNSTRING* functions are not supported under RM-COBOL. It did, however, produce the smallest run-time modules of any of the compilers tested.

Compile time was fairly fast as RM-COBOL did not have to be linked to produce the final output code. I only wish Ryan-McFarland had picked an extension other than .COB for the resulting code. It makes an interesting display when you attempt to edit a file that ends in .COB when you thought it was source code from another compiler.

Very easy to use, RM-COBOL has a fair amount of source code and application packages written in it already on the market. The fact that the *SORT* option is not supported surprised me a great deal when I went to use it.

Although available from third-party vendors, the price seems a little high when you consider that most business applications will require the use of the *SORT* feature at some point in time. The lack of the *STRING/UNSTRING* verb is a minor point since I'm not even sure when you would use the instruction, especially when you consider that this is not an available mainframe instruction.

WATCOM COBOL v. 2.1

Although all the tables for this compiler show N/C (Not Compiled), WATCOM's COBOL version 2.1 is still a good compiler. It is rather different than all the rest in that WATCOM did not implement standard COBOL coding form convention.

For this compiler, comments (asterisks) must appear on column 1. Column 1 is also the continuation column. Margin -A- starts in column 2 and margin -B- starts in column 6. This proceeded to give me fits since all the benchmark programs are set up for standard COBOL margins as being column 8 and column 12, with the comment or continuation column being column 7.

The programs that come with this compiler are excellent for the teaching environment, which is where this compiler is intended to be used. For educational institutes or large organizations involved in training, this compiler would be a good choice since you can make unlimited copies under the single license. If you are a commercial institution you get to pay $3,000 a year for the use of the compiler.

The licensing fee is a yearly amount that must be paid. This compiler has a built-in expiration date that will prevent the compiler from functioning 30 days after the license expires, unless you have paid for the license extension.

WATCOM's compiler is intended to be run in close association with a VAX or similar mainframe since it supports network-type features that allow remote access to host systems. The documentation and sample programs are all keyed to teach the user COBOL and would make an ideal lesson plan for the instructor teaching COBOL. The only flaw I see is rejection of the COBOL margin convention.

Royalty information

Microsoft, Realia, and Nevada COBOLs carry no royalty fees whatsoever. You can freely distribute the run-time modules with the compiled code. Microsoft does require that you sign a licensing agreement and will then give you examples of the copyright notice that must appear at the start of your program.

mbp's royalty agreement states that under version 7.4 the first 50 installations of any system developed with its compiler are included in the purchase price. But with version 9.0, you can only distribute to the first 25 installations and, if you wish to purchase the unlimited run-time license, it will cost you $2,500 instead of the $1,000 it did under version 7.4.

Ryan-McFarland's royalty scheme is based on the number of distributions you make of the run time. For one to 24 it will cost you $100; 25 to 99, $75; 100 to 499, $50; 500 to 999, $35; and 1,000 to 4,999, $25.

Micro Focus has adopted a somewhat easier royalty method. Under Micro Focus's agreement it will cost you a flat $40 for each run time you distribute, regardless whether it is for the level II or Professional COBOL compiler. Micro Focus now refers to this as its application support modules and not as a run-time package since it is only required for screen and file I/O.

Since DRI's products are basically Micro Focus items, I can only guess that their royalty policies are along similar lines.

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

The Tools You Need To C You Thru.

Now the *WizardWare*™ Applications Programmers Toolkit provides everything you need to increase your C programming productivity.

APT™ features include:

- COMPLETE SOURCE CODE (over 5000 lines!)
- File handling with direct & keyed access
- Screen and Report Generators, with full screen handling for your programs
- Generic Terminal Driver for portable code
- String math functions, and string manipulation routines
- Reference Manual on Disk (over 50 pages)
- Tutorial Manual (over 25 pages) with Source for Mailing List Manager
- A host of useful Utilities, Database and File Editors
- Available for Lattice C, Mark Williams C, DeSmet C, BDS C, others.

Also Available: C-STARTER Toolkit, great for learning C!! Includes: Customized APT, DeSmet C Compiler, and "Programming in C on the IBM-PC" (200 pages)

APT/MS-DOS versions	$495
APT/DeSmet C version	$395
APT/BDS C version	$395
C-Starter (binary APT, DeSmet Compiler and Book)	$295
APT/Manual only	$ 50

Detailed Brochures on request

*Manual Cost will be applied if APT purchased within 30 days ($10 re-stocking charge.) U.S. funds only, please.

Trademarks: MS-DOS/Microsoft, Lattice C/Lattice, Inc., MWC86/Mark Williams Co., DeSmet C/C Ware, CI-C86/Computer Innovations, Inc., BDS C/BD Software, DR-C/Digital Research, WizardWare, APT, C-Starter/Shaw☆American Technologies.

Call (502) 583-5527

Ask for APT™ or C-Starter, or Send Check to:

Shaw☆American Technologies

WizardWare™

830 South Second St. · Box 648
Louisville, KY 40201, USA

(C.O.D. and Foreign Orders - Add $5 Shipping/Handling)

References: Bank of Louisville, Citizens Fidelity Bank, Louisville Chamber of Commerce

CIRCLE 34 ON READER SERVICE CARD

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

CROSS COMPILER FOR THE 8086™ FAMILY

LATTICE® C CROSS COMPILER

Subject: C Cross Compiler for the 8086 Family.

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

Features:

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

Price:

VAX (UNIX or VMS)	*$5000*
MC68000	*3000*

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

OEM terms available.
Mastercard/Visa accepted.

Trademarks of: Lattice, Lattice, Inc.; VAX and VMS, Digital Equipment Corp. UNIX, AT&T Bell Laboratories. IBM-PC, International Business Machines. MS-DOS, Microsoft. MC68000, Motorola. 8086/8087, Intel.

UniPress Software
Your Leading Source for UNIX Software.

CIRCLE 81 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. The interpreter can directly load program files created with any ASCII text 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.

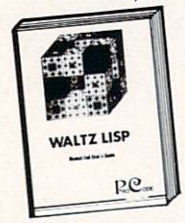
Full Lisp. Functions of type *lambda (expr)*, *nlambda (fexpr)*, *lexpr*, *macro*. Splicing and non-splicing character macros. Full suite of mappers, iterators, etc. Long integers (up to 611 digits). Fast list sorting using user defined comparison predicates. Built-in prettyprinting and formatting facilities. 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. 23
In Oregon and outside USA call 1-503-684-3000

ProCode™ INTERNATIONAL
15930 SW Colony Pl.,
Portland, OR 97224

CIRCLE 60 ON READER SERVICE CARD

What criteria to use

As you can tell from the benchmarks, COBOL happens to be one language where benchmarks do not necessarily give you a true picture of the compilers.

Remember, the compiler you pick should be as closely as possible suited to your purpose. If you need fast sorting speed, mbp, Microsoft, and Micro Focus COBOLs are prime contenders. If sorting is not going to be a requirement, then RM-COBOL will serve your purposes although you get a better deal by purchasing the latest version of Microsoft's COBOL.

One thing that has cropped up in several of the tables is the mention of network support. I believe that this will be the next hot button for both software and hardware, and it will be a major revision to many vendors' way of thinking. Currently, Microsoft, mbp, Realia, and Micro Focus COBOLs all either have run times that support network or multiuser environments or will have them ready by the time you read this. Most vendors will support the Novell network first and then add other networks as they become known. Use of these network run times will usually result in higher costs, as is the case with mbp and Micro Focus COBOLs.

Network support and compiler enhancements that increase productivity are going to be the next changes you see to COBOL because it is such a verbose language. Anything done to improve the number of lines of code a programmer can generate in these compilers will result in that compiler becoming the predominant force the rest of the market must deal with.

I hope you have finished this wrap-up with a better understanding of what the various compilers support and now have a tool to decide which one is just right for you. ∎

Charles Ballinger is a systems analyst and has been involved in computers and programming for a little over 10 years.

Manufacturer	**Product**
Digital Research Inc. Box DRI Monterey, Calif. 93942 (408) 649-3896	Level II v. 3.0
Ellis Computing Inc. 3917 Noriega St. San Francisco, Calif. 94122 (415) 753-0186	Nevada COBOL v. 2.1
mbp Software & Systems Technology Inc. 7700 Edgewater Dr. Ste. 360 Oakland, Calif. 94621 (415) 632-1555	mbp COBOL v. 7.40, v. 9.0
Micro Focus Inc. 2465 East Bayshore Rd. Ste. 400 Palo Alto, Calif. 94303 (415) 856-4161 West Coast (215) 668-2278 East Coast	Professional v. 1.0, Level II v. 2.6.2, VS-COBOL v. 1.0
Microsoft Corp. 10700 Northup Way Box 97200 Bellevue, Wash. 98009 (206) 828-8080	Microsoft COBOL v. 1.12, v. 2.0
Realia Inc. 10 S. Riverside Plaza Chicago, Ill. 60606 (312) 346-0642	Realia COBOL v. 1.0
Ryan-McFarland 609 Deep Valley Dr. Rolling Hills Estates, Calif. 90274 (213) 541-4828	RM-COBOL v. 2.0B
WATCOM Products 415 Phillip St. Waterloo, Ont. Canada N2L 3X2 (519) 886-3700	WATCOM v. 2.1

YOUR CODE MAY BE WASTING ITS TIME!
THE PROFILER™ CAN HELP . . .

- Statistical Execution Profiler
- Works with any language
- Completely configurable
- Up to 16 partitions in RAM/ROM
- Time critical code optimization
- Abnormal code behavior tracking
- Graphic presentation of results
- Easy to use menu interface

THE PROFILER is a software package which gives you, the programmer, a powerful tool for locating time consuming functions in your code and allows you to performance tune your program. With **THE PROFILER** you can determine where to optimize your code for maximum benefit, then measure the results of your efforts.

Using **THE PROFILER,** you can answer questions like:
Where is my program spending its time?
Why is my program so slow? What is it doing?
Is my progam I/O bound? CPU bound? Are data buffers large enough?
How much improvement did my changes make?

THE PROFILER is completely software based and consists of a system resident driver and a monitor program. The memory partitions can range from 1 byte to 1 megabyte in size and can be anywhere in the address space.

NO ADDITIONAL HARDWARE IS REQUIRED!

Requires an IBM PC or compatible system with a minimum 64k and one drive.

THE PROFILER is available for $125.00 **(New Low Price)** from DWB Associates or ask your software dealer. To order or for more information, call or write DWB Associates. VISA/MC accepted. Dealers welcome.

dwb Associates
P.O. Box 5777
Beaverton, OR 97006
(503)629-9645

IBM is a trademark of IBM Corp. MSDOS is a trademark of Microsoft Corp.
THE PROFILER is a trademark of DWB Associates.

CIRCLE 21 ON READER SERVICE CARD

ADVERTISE
in the July issue of
COMPUTER LANGUAGE
Artificial intelligence/ Expert systems

Reservation deadline: May 6

Contact:
Carl Landau
COMPUTER LANGUAGE
131 Townsend St.
San Francisco, Calif. 94107
(415) 957-9353

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 87 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

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

RTL

RELOCATABLE THREADED LANGUAGE

RTL is similar to FORTH but contains features not normally included in threaded languages. Since each word is independently relocatable, RTL allows deleting either individual words or all unused words with garbage collection. All code is romable. The decompiler allows editing directly from RAM. Any word may be redefined retroactively. It supports string variables, local variables, bitmaps and pictured numeric formating.

RTL is currently available for the 68000, 6809, 8086 and Z80. The 68000 version is a 32 bit implimentation.

RTL Programming Aids $150
10844 Deerwood SE
Lowell, MI 49331 INCUDES ALL
(616) 897-5672 SOURCE CODE

CIRCLE 71 ON READER SERVICE CARD

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

For more information call (617) 641-2379
WSS
Wizard Systems Software, Inc.
11 Willow Ct., Arlington, MA 02174
Visa/Mastercard accepted

CIRCLE 86 ON READER SERVICE CARD

C on the Macintosh

By Michael Rothman

What's the most interesting thing about C compilers for the Apple Macintosh? Not surprisingly, it's the same thing that's interesting about the Mac—the unique Mac user interface.

The nontechnical user has two relatively easy questions to answer: do I understand this interface (desktop, mouse, graphics), and do I like it? The technical user (or the company trying to sell a technical application) has to struggle with a more complicated reality: what's best for the user may not be best for the developer. Is the Finder, the Macintosh operating system, an appropriate environment for development?

The companies who make the five Macintosh compilers I review here have all had to answer this question. Some have said yes (Consulair, Softworks) and piggybacked onto Apple's own Macintosh Development System (MDS), which uses all the special Mac features. One has said a resounding no (Manx) and essentially provided its own operating system. And some have compromised, trying to have the best of both worlds (Megamax, Hippopotamus).

Is the Mac a good development environment? This is what a colleague of mine calls a "religious" question, and I won't try to answer it here. But this religious question makes it crucial that you don't take the benchmarks as gospel! You should also consider how you feel about the interface and whether you prefer to have an underlying development system from Apple or from some other manufacturer. (For more on this subject, please see the two sidebars.)

Whatever the companies who make the compilers think of the Mac as a development system, they all think it's important to support the unique Mac interface in your environment, and they all make access to the Mac ROM toolbox as simple as a function call. The compilers straighten out the differences between C and Pascal calling interfaces—the difference between C and Pascal strings is either handled directly by compiler-provided "glue" routines or the user must call a conversion function.

Table 1 provides basic information on the Macintosh compilers. Table 2 lists the benchmark results. For a discussion of three of the benchmarks used—Sieve of Eratosthenes (which generates prime numbers, Fib (which generates Fibonacci numbers), and Matrix (which tests floating point arithmetic speed)—see the C review in the February issue of *COMPUTER LANGUAGE* (p. 82).

A fourth benchmark, Macptr, examines the efficiency of pointer dereferencing. This is the deref.c benchmark used in the February issue, but it has been modified to initialize each level of the pointer reference to a real RAM value. In the table, the time to do this initialization is included in the "Run load" category rather than the "Run" category.

Table 3 shows the results for the Sieve benchmark in greater detail.

Consulair Mac C

Consulair's Mac C includes the compiler and a prerelease version of Apple's MDS including the editor, assembler, and linker. The system also includes a Consulair version of the Exec (batch file) utility, meant to substitute for the Apple version until it is available. If you pay extra, you also get the Mac C toolkit, a large library of useful functions that Consulair has "prewritten" for you. This should not be confused with the Macintosh's own ROM toolbox, which Consulair supports.

Consulair does not support floating point arithmetic but plans to add it in an update. Ditto for register variables.

The Mac C system includes sources for all the libraries. The standard C library comes in three pieces.

The documentation for the system is a manual of about 100 pages, similar in shape and binding to the manuals Apple provides with the Mac. Thus it lies flat and fits in a relatively cramped workspace, a nice touch. Contents include an overview of the C language (useful mainly for its description of machine-specific implementation details), instructions on running the system, an unusual and interesting section on the compiler's strategies for code generation, a description of the standard library functions, and a section on Consulair's Mac C toolkit. The manual has no index.

The compiler runs as described in the sidebar "MDS-based systems." Among the compilers that run from the Finder, Consulair has the best selection of compilation options (intersperse C source in assembler output, verbose errors, choose integer size of 16 or 32 bits, and several others).

I had to remove the *void* type reference in the Fib benchmark. With that change, everything compiled very easily. Incidentally, the Consulair error messages I encountered were more complete than the average and made intelligent guesses about what was wrong in ambiguous situations.

Of course, one of the strengths of C is that it can be a very ambiguous language. The error messages for Pascal, for instance, are usually much more precise and have a certain authoritarian manner. In the late 1970s, an implementation of Pascal on the Yale Dec-20 insulted you after you reached a certain error count.

Hippopotamus Software Hippo-C

I would be tempted to buy Hippo-C on name alone: there's not enough whimsy in this industry. (Manx's Aztec C shares the virtue of an unusual name.) Hippo-C comes at two levels. Level 1 is designed for the more casual user and level 2 for the serious developer. I didn't rigorously test level 1, but the company says its benchmarks would be less impressive than those for level 2.

Level 2 was only available in a prerelease form and only at the last minute, so I didn't have time to check every feature. Level 1 produces a pseudocode that is interpreted at run time; level 2 produces assembly source that is assembled and linked. There is no way to produce a stand-alone application in level 1, but you can do it in level 2. The benchmarks are level 2. Some of the features described here are currently only in level 1, but Hippopotamus says that eventually level 2 will be a superset of level 1.

THE PROGRAMMER'S SHOP™

helps compare, evaluate, find products. Straight answers for serious programmers.

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 Literature - Compare Products

Evaluate products Compare competitors. Learn about new alternatives. One *free call* brings information on just about any programming need. Ask for any ***"Packet"*** or "Addon Packet": ☐ ADA, Modula ☐ "AI" ☐ BASIC ☐ "C" ☐ COBOL ☐ Editors ☐ FORTH ☐ FORTRAN ☐ PASCAL ☐ UNIX/PC or ☐ Debuggers, Linkers, etc.

RECENT DISCOVERIES

FASTER C - Lattice users eliminate Link step. Normal 27 seconds, Faster C in 13 secs. MSDOS $95

"C" LANGUAGE

	OUR PRICE
MSDOS: C86-8087, reliable	call
Instant C - Inter., fast, full	495
Lattice 2.1 - improved	call
Microsoft C 2.x	329
Williams, debugger, fast	call
C Systems & debugger	175
CPM80: Ecosoft C-now solid, full	225
BDS C - solid value	125
MACINTOSH: Softworks	365
Megamax-object, full	295
Consulair's MAC C	295

Compare, evaluate, consider other Cs

BASIC

	RUNS ON	
Active Trace-debug	86/80	75
BASCOM-86 - MicroSoft	8086	279
BASIC Dev't System	PCDOS	115
BetterBASIC - 640K	PCDOS	185
CB-86 - DRI	CPM86	419
Prof. BASIC Compiler	PCDOS	89
Databurst - screens	MSDOS	215
SCREEN SCULPTOR	PCDOS	115

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
MINCE-like EMACS	PC/80	149
PMATE-powerful	8086	185
VEDIT-full, liked	86/80	119

UNIX PC

COHERENT - for "C" users	PClike	475
COHERENT-NCI-Realtime	PClike	call
XENIX - plus C to MSDOS	PC	1275

Ask about run-times, applications, DOS compatibility, other alternatives. UNIX is a trademark of Bell Labs

LANGUAGE LIBRARIES

GRAPHICS: GraphiC-source in C	MSDOS	250
GRAPHMATIC-3D. FTN, PAS	PCDOS	125
HALO-fast, full-all lang.	PCDOS	145
FILE MGNT: BTrieve-all lang.	MSDOS	215
CIndex + -source, no royal.	86/80	375
CTree-source, no royal.	ALL	375
dBC ISAM by Lattice	8086	235
dB VISTA-"Network" Structure	MSDOS	465
PHACT-up under UNIX, addons	MSDOS	225
OTHER: CUtil by Essential	MSDOS	139
Greenleaf - 200 +	MSDOS	149
CSharp - **Real-Time**	MSDOS	600
PORTABLE C to PC, Mac, II	Many	125
SOFT Horizons - Blocks I	PCDOS	139
SCREEN: CURSES by Lattice	PCDOS	125
CView - input, validate	PCDOS	195
MetaWINDOW - icons, clip	PCDOS	139
PANEL - many lang, term	MSDOS	265
ProScreen - windows, source	PCDOS	415
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

Assembler & Tools - DRI	8086	159
Atron Debugger for Lattice	PCDOS	395
cEnglish - dBase to C	MSDOS	750
C Helper: DIFF, xref, more	86/80	135
CODESMITH-86 - debug	PCDOS	139
MacASM-full, fast, tools	MAC	115
MBP Cobol-86 - fast	8086	680
METAWINDOW - graph, fonts, clip	PCDOS	135
Micro: SubMATH-FORTRAN full	86/80	250
Microsoft MASM-86	MSDOS	125
MSD Debugger	PCDOS	119
Multilink - Multitasking	PCDOS	265
PC/FORTH + -well liked	MSDOS	219
PFIX-86 Debugger	MSDOS	169
PL/1-86	8086	495
Polylibrarian - thorough	MSDOS	95
PolyMAKE	PCDOS	95
PROFILER - flexible	MSDOS	125
Prolog-86-Learn, Experiment	MSDOS	125
SLK/F - Copy Protection	PCDOS	145
SYMD debugger-symbols	PCDOS	119
TRACE86 debugger ASM	MSDOS	115

Note: All prices subject to change without notice. Mention this ad. Some prices are specials. Ask about COD and **POs.** All formats available.

Call for a catalog, literature, and *solid value*

800-421-8006

THE PROGRAMMER'S SHOP™

128-L Rockland Street, Hanover, MA 02339

Visa Mass: 800-442-8070 or 617-826-7531 MasterCard 8517

CIRCLE 70 ON READER SERVICE CARD

C Helper™

FIRST-AID FOR C PROGRAMS

Save time and frustration when analyzing and manipulating C programs. Use C HELPER's UNIX-like utilities which include:

DIFF and **CMP** – for "intelligent" file comparisons.
XREF – cross references variables by function and line.
C Flow Chart – shows what functions call each other.
C Beautifier – make source more regular and readable.
GREP – search for sophisticated patterns in text.

There are several other utilities that help with converting from one C compiler to another and with printing programs.

C Helper is written in portable C and includes both full source code and executable files for $135 for MS-DOS, IBM AT CPM-80 or CPM-86. Use VISA, Master Card or COD.

Call: **800-821-2492**

Solution Systems™
335-L Washington Street
Norwell, MA 02061
617-659-1571

CIRCLE 91 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 dBASE II "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 established in "Programming in Prolog."

CONTEST: Win $1,000. Ask about it. Deadline of 4/30/85.

AVAILABILITY: PROLOG-86 runs on MSDOS, PCDOS, IBM AT or CPM-86 machines. We provide most formats. The price of PROLOG-86 is **only $125.**

Full Refund if not satisfied during first 30 days.
800-821-2492

Solution Systems™
335-L Washington Street
Norwell, MA 02061
617-659-1571

CIRCLE 92 ON READER SERVICE CARD

Like Manx's Aztec C, Hippo-C implements a command shell (called the Hippo operating system or HOS) to make the Mac more like a conventional computer. But Hippo-C allows command entry either through a menu or keyboard entry. The commands include the all-time UNIX hits like *ls, mv, rm*, and so forth. Command line redirection and the *argc argv* conventions are also supported.

Hippo-C has its own editor, Ed, which is similar to Edit with some C development specific features. One of these is a *goto* line number command, which is great for error diagnosis. Files prepared under a standard Mac text editor work fine in Hippo-C, but a file created under Hippo-C has different line ending conventions and so will not show correctly in a standard Mac editor.

Hippo-C has a feature unique among the compilers I tested—level 1 has a C source level debugger. (Hippopotamus says it will eventually include one in level 2, but not in the first release.) I didn't test the utility carefully, but assuming everything works well, this could be a tremendously useful developer's tool.

Hippo-C also has a make file utility which, given the name of a C file, creates a batch file to compile, assemble, and link it.

Hippo-C might be a good system for you if you're interested in how it's done—the source for the HOS UNIX-like commands is included. So is the source for the standard C libraries. The archiver doubles as a librarian. The documentation (level 1) is contained in a well-written, book-size three-ring binder.

The system seems to be K&R compatible; however, there was no floating point support in my prerelease level 2 so I skipped the matrix benchmark. (Hippopotamus says that eventually there will be C-integrated floating point support but currently the way to go is to use the Mac ROM routines.)

I had to remove the *void* type and include stdio.h in everything because *exit()* is defined in stdio.h and Hippo-C's own code references *exit*. *Myref* revealed a problem in the prerelease level 2. When I ran the benchmark it whizzed by so fast that I got suspicious and inserted a *printf* statement into the loop that contains the actual dereferencing. Sure enough, the *printf* statement failed to appear when I ran the new version—the loop was never executing. Changing the counter variable to a *long* instead of an *unsigned* cleared up the problem.

My copy of Hippo-C level 2 was clearly unfinished, but after all, that's exactly what I had been told. Judging by level 1, the bugs will be out by release time (are you listening Hippopotamus?).

Manx Aztec C

Manx has taken the most radical approach to the Mac interface issue by providing Aztec C with its own command shell (called, sensibly enough, the shell). No menus, no mice, no desktop—just a good old-fashioned, user-hostile computer interface. You'll think AT&T snuck into your Mac one night. The Aztec C shell works very well, mimicing many important UNIX commands, emulating a hierarchical file structure, and also providing a simple shell script (batch file) capability.

Along with the shell and compiler, Aztec C comes with its own macro assembler and a linker. Because of the shell, these are used in a manner very similar to that of typical (non-Mac) C environments. For example, a compile of the Sieve program with no options is invoked by typing "cc sieve".

The developer has a choice of writing code either for the shell or for the standard Mac Finder. Manx achieves this by offering three different application start-up routines. Depending upon which one you link in, your application will have greater or lesser UNIX-like capabilities. Note that Aztec C can support the *argc argv* command passing conventions for programs designed to run under the shell.

Aztec C supports most of the Mac ROM toolbox routines. Manx says they will be adding toolbox routines with each update.

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

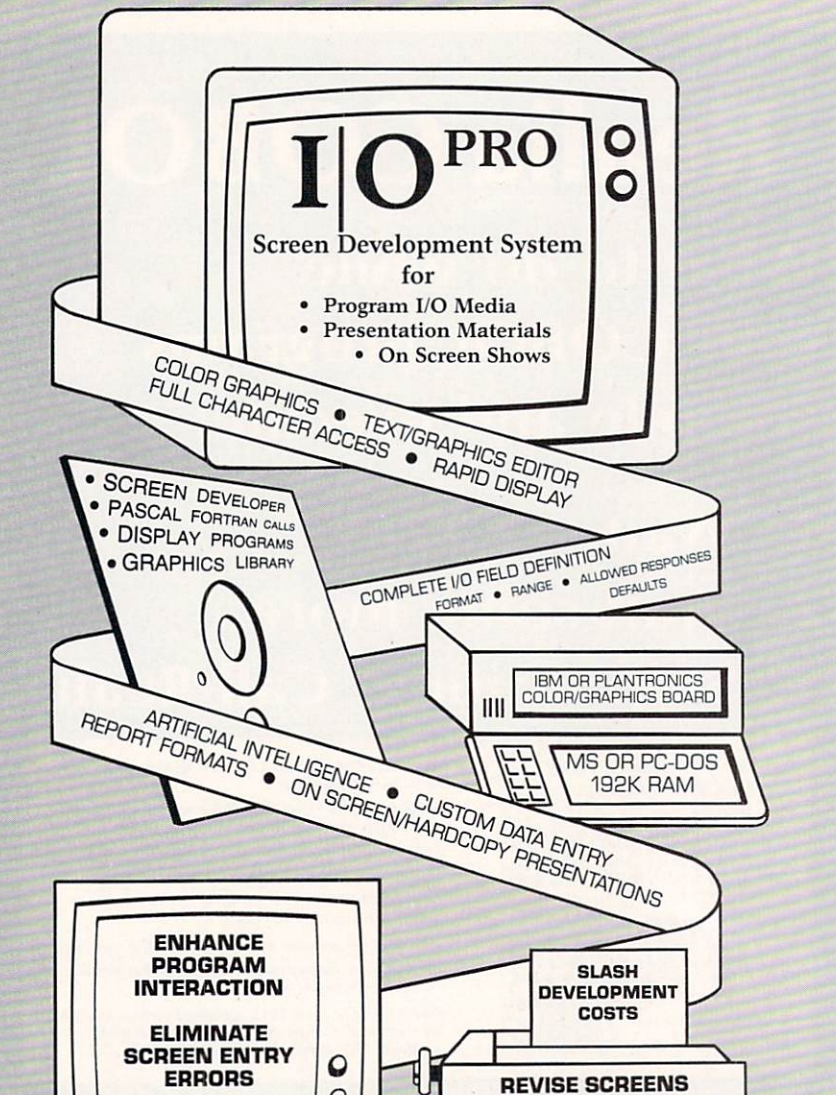
Thunder Software

• **The THUNDER C Compiler** - Operates under the APPLE Pascal 1.1 operating system. Create fast native 6502 programs to run as stand alone programs or as subroutines to Pascal programs. A major subset of the C defined by K & R. Includes a 24 page users guide, newsletters. Macro preprocessor, runs on APPLE][,][+, //e, //c. Source code for libraries is included. **Only $49.95**
• **ASSYST: The Assembler System** - A complete 6502 editor/assembler and lister for APPLE DOS3.3. Menu driven, excellent error trapping, 24 p. users guide, demo programs, source code for all programs! Great for beginners. **Only $23.50**
• **THUNDER XREF** - A cross reference utility for APPLE Pascal 1.1. XREF generates cross references for each procedure. Source code and documentation provided. **Only $19.95**

Thunder Software POB 31501 Houston Tx 77231 713-728-5501
Include $3.00 shipping. COD, VISA and MASTERCARD accepted

CIRCLE 65 ON READER SERVICE CARD

I/O A BORE? NOT ANY MORE!


I|O PRO
Screen Development System
for
• Program I/O Media
• Presentation Materials
• On Screen Shows
COLOR GRAPHICS • TEXT/GRAPHICS EDITOR
FULL CHARACTER ACCESS • RAPID DISPLAY
• SCREEN DEVELOPER
• PASCAL FORTRAN CALLS
• DISPLAY PROGRAMS
• GRAPHICS LIBRARY
COMPLETE I/O FIELD DEFINITION
FORMAT • RANGE • ALLOWED RESPONSES
DEFAULTS
IBM OR PLANTRONICS COLOR/GRAPHICS BOARD
MS OR PC-DOS 192K RAM
ARTIFICIAL INTELLIGENCE • CUSTOM DATA ENTRY
REPORT FORMATS • ON SCREEN/HARDCOPY PRESENTATIONS
ENHANCE PROGRAM INTERACTION
ELIMINATE SCREEN ENTRY ERRORS
SLASH DEVELOPMENT COSTS
REVISE SCREENS IN SECONDS

M|E|F Environmental, Inc.
P.O. Box 26537
Austin, TX 78755 (512) 251-5543

$450 : OEM Pricing and Licenses Available
Demonstration Diskette $10
applicable to purchase

CIRCLE 15 ON READER SERVICE CARD

Library handling is very flexible; a good library utility is included that lets you create and modify your own libraries. In addition, since the source for all the standard libraries is included, it is possible to customize your environment. The linker selects referenced modules from libraries, leaving unreferenced code alone (unlike the MDS-based systems).

The Aztec C compiler comes in two versions. The more expensive version includes a number of utilities familiar to UNIX fans, such as make, grep and diff and, perhaps most amazing, a clone of the VI editor, which Manx calls the Z editor.

Like several other companies, Manx also has an agreement with Apple to provide RMaker, Edit, and MacsBug. This means you have your choice of a mouse-based or a traditional editor. Manx also throws in a RAM disk for the fortunate 512K owner. This can of course substantially improve development speed. (It was not used for the benchmarks.)

Manx documentation is excellent—two

Where does the time go?

The most difficult part of preparing this review was attempting meaningful timing comparisons among the compilers. The question seems pretty straightforward: How long does it take to run a compile and a link of a given program with this system? Not surprisingly, it is the innovative Mac environment that makes it difficult to answer the question.

All but two of these compilers (Manx's Aztec C and Hippo-C) run under the Finder, the Macintosh operating system. The Finder maintains much more information than a typical operating system and trips through it between development steps (for example, compilation and assembly, assembly and link) can significantly lengthen the process. Thus, each of the compilers reviewed here offer at least one and sometimes several tools as short cuts. (These are outlined in the sidebar "MDS-based systems" and in the individual reviews.)

Unfortunately, these tools vary from compiler to compiler, and this makes meaningful comparison difficult. It's equally difficult to just ignore the tools in question since in many cases they are the default mode and operate automatically. To disable them and test in that mode would give an unrealistic account of a typical user's experience.

large three-ring binders with tutorials, full descriptions of all utilities, descriptions of each library function, and more. Only an index is lacking.

K&R is fully supported, except for the bitfield data type. Scanning the benchmarks, it's hard to resist the conclusion that Manx blows away the competition for pure compilation speed. Unburdened by trips in and out of the Finder and helped by the Shell's clean handling of two disks, the total compilation and link time in every benchmark is at least 40% less than the nearest competitor. Manx also does well in code size and run time. The compiler could only handle six levels for each pointer in the dereferencing benchmark.

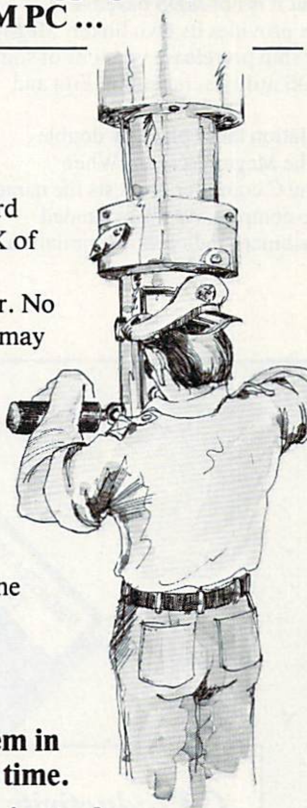
I found the shell a pleasure to work in (but be warned, I'm an old UNIX hacker!) and the illusion of a more conventional computer is complete. All in all an excellent effort.

With a great deal of trepidation, I chose to run each compiler in its most "natural" mode, and this produced the times in the Table 2. By natural mode I mean that I used any shortcuts that presented themselves automatically as part of the run process. I did not use tools (such as Aztec C's RAM disk or Megamax's optimizer) that require separate initialization or run as an additional step in compilation. I did not count time for user-dependent actions, such as typing in a file name to a dialogue box.

In addition, I have provided for the Sieve benchmark a more elaborate table showing each step in the process for each compiler. This gives the reader an idea where the time goes. The wise reader will bear in mind that steps involving the Finder are not a fixed length since they depend on the complexity of the Mac desktop and other factors. Furthermore, as new tools become available (for example, better batch utilities) and the MDS system is improved, these compilers may improve their performance. If time to compile is all that is holding you back from a particular compiler, you would do well to call the manufacturer and inquire for the latest information.

Debugging Bugging You?

Torpedo program crashes and debugging delays with debugging dynamite for the IBM PC ...



UP PERISCOPE!

First, you install the hardware.

The hardware's a special memory board that fits in a PC expansion slot. Its 16K of write-protected memory contains Periscope's resident symbolic debugger. No runaway program, however berserk it may be, can touch this memory!

Then you UP PERISCOPE.

Use Periscope's push-button break-out switch to interrupt a running program ... even when the system's hung! Periscope supports Assembly, BASIC, C and Pascal. In addition to the usual debugging capabilities, some of Periscope's features are:

Stop your system in its tracks at any time.

Use symbol names instead of addresses.

Run a program on one monitor and debug on another.

Monitor your program's execution with Periscope's comprehensive breakpoints.

Debug memory-resident programs.

Put your time to better use.

The Periscope system is $295. It carries a 30-day money-back guarantee and includes the memory board, remote break-out switch, debugger software, 100-page manual, and quick-reference card. The memory board is warranted for one year. A demonstration disk is $5.00.

System requirements for Periscope are an IBM PC, XT, AT or Compaq, PC-DOS, 64K RAM, 1 disk drive and an 80-column monitor. For MasterCard and Visa orders only, call 800/421-5300 (ext. R96) 24 hours a day. For additional information, call 404/256-3860 from 9 AM to 5 PM Eastern Time.

Get your programs up and running;

UP PERISCOPE!

Data Base Decisions / 14 Bonnie Lane /Atlanta, GA 30328

CIRCLE 30 ON READER SERVICE CARD

Megamax C

The Megamax C compiler is unique among those tested in that it produces linkable object modules directly from its compiler without an intervening assembly step. The system operates from the Finder, but it is not MDS based—Megamax provides its own linker. Megamax does ship prerelease versions of some of the MDS utilities, including Edit and RMaker.

Compilation takes place by double-clicking the Megamax icon. When loaded, the C compiler requests the name of a file to compile. As the expanded Sieve benchmark indicates, compilation time (excluding loading of the compiler from the Finder) is very fast. The first time I ran Megamax, the compiler came back so fast that I thought compilation had failed, and I recompiled several times before I figured out what was happening!

The linker works similarly (but more slowly), taking a list of files.

Megamax supports in-line assembly, an important feature since there is no assembler associated with the system. Megamax also supports overlays—in fact, applications compiled under Megamax always have at least two code segments: an initialization and a main (you don't have to do anything to set this up). The initialization segment runs and then is replaced by the main. (Incidentally, the code sizes in the chart combine both segments. In our benchmarks the initialization segment was small, averaging 150 bytes.)

Besides the utilities already mentioned, Megamax includes a disassembler (currently undocumented), which is useful since there is no other easy way to see what kind of code the compiler produced. Most intriguing of all is a code optimizer called mmimp. I tried it on Matrix and got an improvement of only 40 bytes, but in a larger application it might well be more useful.

Megamax has a librarian for creating, listing, and appending to libraries. The compiler supports the Mac ROM routines. Documentation is good. A well-organized three-ring binder includes instructions, descriptions of the library functions, full technical documentation on library and object module formats, and machine dependencies. The library sources themselves are not included. K&R seems to be fully supported.

I had to remove the *void* type from Fib and include stdio.h in Matrix (it contains the definitions of *getchar* and *putchar*). After that everything compiled easily. Like several of the compilers, Megamax couldn't interpret the bell character and printed instead the ubiquitous Macintosh box character.

Softworks C

Softworks C is based largely on Whitesmiths C, whose implementation has been around for a number of years on a wide variety of systems. It runs under the Finder and produces assembly code for the Apple MDS. The MDS prerelease system is shipped with the compiler.

Libraries are provided as several large object files. Whitesmiths won't let Softworks provide source for its libraries, so only Mac-specific source is provided.

Documentation consists of a machine-independent Whitesmiths C manual, the MDS documentation, and four sheets on the compiler. The Whitesmiths manual is not very useful since it mainly describes the functions based on UNIX version 6, while Softworks has wisely chosen to support the version 7 functions, familiar to K&R readers.

You will note in the benchmarks that Softworks code is large. Softworks shares with all the MDS-based systems the problem that "dead" code is not detected by the linker. That is, if you link in an object module containing 50 routines, you are stuck with all that code even if your application only calls one of the routines.

The problem is exacerbated in Softworks C, however, since the libraries provided are quite large and the user can

C Productivity Series—
The Professional's Edge

Blaise Computing has a range of programming aids for the most popular C compilers in the IBM environment that no serious system developer should be without. These packages help you to easily access advanced capabilities of the hardware and operating system, and to finish your projects with a substantial saving of time and effort. With software development costs and pressures as great as they are, can you afford not to take advantage of the finest tools available?

- **C TOOLS**™ puts advanced string handling functions at your disposal and provides a high-level interface to all BIOS functions from your C program. Complete screen handling, graphics primitives, and a substantial group of useful, general-purpose functions are also featured. **$125**
- **C TOOLS 2**™ lets your program perform all the advanced DOS 2.0 services. Program chaining, software interrupt handling, and dynamic memory allocation are all done "right." Buffer and file handling functions are provided, as well as a general DOS gate. **$100**
- **C VIEW MANAGER**™ is our display screen management system that makes screen development and documentation much faster. It comes with a complete library of C functions which use the screens you have developed to recall and display information, capture and validate field data entry, and provide context-relevant help files. **$275**
- **ASYNCH MANAGER**™ is a library of interrupt-driven routines providing a general interface to both COM ports for your asynchronous communications applications. Introductory price of **$175** includes all source.

All of these products may be used by developers with no royalty payments to Blaise Computing. Source code either comes with the package, or is available. We support Lattice, Computer Innovations, and Microsoft C compilers. To expedite your order or to obtain further information, call or write us directly.

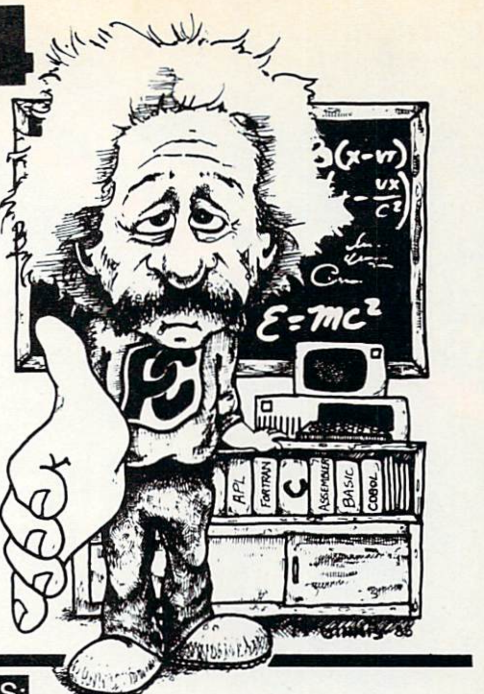
Blaise Computing's Programmer Productivity series is also available in versions for the Pascal language.

BLAISE COMPUTING INC.
2034 Blake Street Berkeley, CA 94704
(415) 540-5441

CIRCLE 8 ON READER SERVICE CARD

ADVANTAGE #4

At Programmer's Connection we listen to programmers and we take the time to research and test our products. We are confident in our knowledge of the software market and that's why we welcome your inquiries. Our goal is to help you make an informed decision when purchasing a programming language or utility. Call us today — you'll discover the difference. Programmer's Connection will revolutionize the way you think about software development.



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 Continental U.S.

Programmer's Development Tools:

C LANGUAGE:

	List	Ours
Computer Innovations C-86 Compiler	395	**299**
DeSmet C Compiler with Debugger	159	**145**
Lattice C Compiler from Lifeboat	500	**299**
Mac C by Consulair *for MacIntosh*	295	**259**
Mark Williams C Compiler w/Source Debugger	500	**449**
Xenix Development System by SCO	1350	**1099**

Special Combination Offer
Lattice C Compiler and
C-SPRITE Symbolic Debugger
Combined List Price $675 Our Price **$429**

OTHER LANGUAGES:

8088 Assembler w/Z-80 Translator 2500 AD	100	**89**
APL*Plus/PC by STSC	595	**499**
BetterBASIC by Summit Software	200	**169**
Golden Common LISP by Gold Hill	495	**439**
Macro Assembler by Microsoft . *New Release*	150	**119**
Modula-2/86 by Logitech	495	**439**
Professional BASIC by Morgan Computing	95	**89**

C UTILITIES:

Asynch Comm Library by Greenleaf	160	**129**
C Power Paks from Software Horizons	Call	**Call**
C-Sprite *Symbolic Debugger for Lattice*	175	**159**
C Utility Library by Essential Software	149	**119**
dBC *dBase/C Interface* by Lattice	250	**219**
DOS LINK Support *for DeSmet C*	35	**35**
English-to-C/C-to-English by Catalytix	100	**100**
ESP for C by Bellesoft	349	**279**
Graphic C by Scientific Endeavors	195	**169**
Greenleaf C Functions Library	175	**129**
Halo *Graphics* by Media Cybernetics	200	**125**
PANEL *Screen Editor* by Roundhill	295	**234**
Run/C Interpreter by Age of Reason	150	**129**

Introducing **Pre-C** by Phoenix Software
Complete lint-like utility that helps detect logic errors by searching for inconsistencies in functions and data types across multiple files.
List Price $395 Our Price **$339**

C UTILITIES:

Safe C Standalone Interpreter by Catalytix	400	**400**
Safe C Dynamic Profiler by Catalytix	150	**150**
Safe C Runtime Analyzer by Catalytix	400	**400**
Windows For C by Creative Solutions	195	**139**

c-tree by Faircom
Full featured B-Tree functions for high speed ISAM file management. Comes as C source code which can be compiled on almost any system including MacIntosh. *No royalties on generated code.*
List Price $395 Our Price **$359**

OTHER PRODUCTS:

APL2C by Decision Images *Interfaces APL to C*	150	**139**
Btrieve by SoftCraft	250	**199**
Dr. Halo by Media Cybernetics	95	**79**
FORTRAN Libraries by Alpha Comp. Serv.	Call	**Call**
FORTRAN Scientific Subroutine Library	175	**159**
Periscope *Debugger* by Data Base Decisions	295	**269**
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**
Screen Sculptor by Software Bottling	125	**109**
XTC *Text Editor* by Wendin	99	**89**
Xtrieve by SoftCraft . . . *Sale!*	195	**149**

CODESMITH-86 Symbolic Debugger by Visual Age
New version 1.9 provides dual-mode patching assembler, branch-to-patch mode, stop-on-data compare/mis-compare, dual monitor debug mode, breakpoints and passpoints, machine state snapshot and hotline technical support.
List Price $145 Special Price **$119**

Prices are subject to change without notice.
Account is charged when order is shipped.

VISA MasterCard

Call for our new Spring Catalog In Canada:
1-800-336-1166 1-800-225-1166

PC
Programmer's Connection
136 Sunnyside Street
Hartville, Ohio 44632
(216) 877-3781 (In Ohio)

"Programmers Serving Programmers"

CIRCLE 27 ON READER SERVICE CARD

Software Development Tools

If:

Manufacturer-Compatible, Fully-Supported Cost-Effective, Cross and Native Development tools are important to you!

Then:

Let us tell you about MICROTEC RESEARCH's extensive line of field-proven software development tools for serious software developers.

C
Pascal
Assemblers
Simulators

Manufacturer Compatible

MICROTEC RESEARCH has been providing flexible and economical solutions for software developers since 1974. We've grown with the industry as our cross software development tools have transformed many large and small computers into powerful cross development systems for microprocessor applications. Our software tools are manufacturer compatible. Therefore, software made using manufacturers development systems may be more productively used in the MICROTEC RESEARCH cross-development environment.

Fully-Supported

MICROTEC RESEARCH's products are continually maintained and updated based upon the experience of thousands of installations worldwide. Revisions and modifications are distributed to licensees in warranty or covered by a service contract. Upgrades, which are major enhancements to a product's capabilities, are available at a significant discount. Our update/upgrade policy assures users they'll always be current with the fast moving world of microprocessor development.

Effective Differences

Beginning with product concept, through development, quality assurance, and post-sales support — **Quality, Compatibility,** and **Service** are the differences which set MICROTEC RESEARCH apart from the others.

Start Saving Time & Money

If you are a serious software developer, shopping for software development tools, call or write today for more information. **800-551-5554** In CA call **(408) 733-2919**

Target Microprocessors		Host Computers	Software Tools
8086/80186	68000/08/10	DEC VAX, PDP-11	C Compilers
8096	6809	DG MV-Series	Pascal Compilers
8080/8085	6800/01/02	DG Eclipse	Assemblers
8051	6805	Apollo	Linking Loaders
8048	6301	UNIX Systems	Librarians
Z8002	6305	IBM PC	Conversion/ Download
Z80	6500	Data General/ONE	Communications
Z8	G65SCXX, G65SC1XX	HP 150	VT-100 Emulator
NSC 800	R65C00, R6500/1..	DEC Rainbow	
9900...	others	others	

for Serious Software Developers

MICROTEC RESEARCH™

3930 Freedom Circle, Suite 101, Santa Clara, CA 95054
Mailing Address: P.O. Box 60337, Sunnyvale, CA 94088
(408) 733 2919 • Telex (ITT) 4990808

CIRCLE 49 ON READER SERVICE CARD

MDS-based systems

By the time you read this, Apple's long-awaited Macintosh 68000 Development System (MDS) may well be available. The Consulair and the Softworks C compilers reviewed here depend on this system for their assembly and link and each is currently shipped with a prerelease version of it. This practice will be discontinued when the release version is available.

The following is a short description of development on the MDS systems.

You work under the Finder in the conventional Mac environment. The programmer prepares the C source code using Apple's editor, Edit. This is a text-only editor similar to MacWrite without the font and graphics frills and supporting up to four files open at once.

The finished source is compiled by clicking on the chosen company's C compiler, which loads and puts up the usual Mac file menu. The user selects the file to be compiled and compilation begins. The compiler invokes the assembler in case of a successful compile, otherwise it calls up the editor on the offending source and the error listing. The assembler can also be called separately.

The developer needs to prepare for the linker a short text file that lists the modules to be linked and provides for a number of listing options. The linker is invoked similarly to the compiler and assembler. It produces a .MAP file showing the memory location of routines and variables.

The MDS provides two tools that can speed up your work. First, each utility in the system (editor, assembler, linker) has an entry "Transfer" in its menu bar. From the pull-down menu you can select one of the other utilities and bypass a trip through the desktop between, for example, the editor and assembler.

The other timesaver will be the Exec program, which is a batch file utility. This is not in prerelease versions of MDS, but it is a welcome sight on the horizon for desk-bound developers tortuously guiding the Mac through the compile process when they could be grabbing a snack. (Consulair is shipping its own Exec utility until Apple's is ready. I did not test it for this review.)

The MDS also includes a resource compiler called RMaker which can produce resource files from a text file that defines them. (Resource is Mac parlance for data that can be shared among various applications and/or has a logically modular nature. Fonts are perhaps the best known resource.) And the system comes with a fine assembly level debugger.

Incidentally, even the compilers that don't use the MDS assembler and linker are shipped with prerelease versions of some MDS tools, usually including the resource compiler. The particular elements vary: if the entire system is not yet commercially available when you buy your compiler, check with the manufacturer to see which pieces are currently shipped.

Bundled software and essential information

Manufacturer	Name	Price	Version	Editor	Assembler	Linker	Librarian	Source level debugger	No. of combiler phases	Separate optimizer pass	Compiler output	Full K&R	Full K&R storage classes	Compiler source code included
Consulair	Mac C	$425.00	1.0	yes	yes	yes	no	no	3	no	ASM	yes	yes	no
Hippopotamus Software	Hippo-C level 2	$399.95	1.0	yes	yes	yes	yes	no[1]	2	no	ASM	yes	yes	no
Manx	Aztec C 68000 level c	$499.00	1.06C (D avail)	yes (2)	yes	yes	yes	no	1	yes	ASM	yes	yes	no
Megamax	Megamax C	$299.95	1.2	yes	yes	yes	yes	no	1	yes	OBJ	yes	yes	no
Softworks	Softworks C	$395.00	1.0	yes	yes	yes	no	no	3	no	ASM	yes	yes	no

1. Level 1 has a source level debugger.

Table 1.

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

neither recompile smaller ones (there's no source for the standard libraries) nor extract only the desired routines (no library utilities exist in MDS, at least in the prerelease). Softworks supports the Mac ROM routines, and if you write your code using only the ROM for I/O (no C library functions), the executable will be much smaller since you need not link all the libraries. But of course it's not portable C if you do that.

Softworks is K&R compatible with a few quirks. All globals must be in a single file, and all globals must be initialized. Local statics are not supported. You can't initialize a pointer to a function at compile time, and you must use a special function to take a function's address at run time. The last line in a source file must be blank. Any routine that includes stdio.h must also include a file called std.h.

Once I made the changes indicated, the programs compiled without problem. The size of the various utilities makes it almost impossible to go through the entire compilation process without at some point moving a file (either the .ASM or .REL) from one disk to another—in fact, this is what Softworks recommends. The extra time this adds is reflected in the benchmark charts (see especially the expanded Sieve benchmark). Like several compilers, Softworks printed the bell character rather than ringing the bell.

In summary

For those of us who bought a Mac back in the first months of its existence, it is an indescribable pleasure to have all those months of "Real Soon Now" products finally coming through. Developers in particular have had a hard time of it on the Mac with only BASIC to satisfy creative cravings.

Now, in a relatively short time span, we

Benchmark results[1] (sec)

Sieve

Compiler	Compile/ assemble	Link	Total	Run load	Run	Size (bytes)
Consulair	86	66	152	10	10	17,654
Hippo level 2	42	60	102	3	13	30,648
Manx	20	29	49	2	7	13,274
Megamax	21	93	114	8	7	13,816
Softworks	174	107	281	10	9	46,914

Fib

Compiler	Compile/ assemble	Link	Total	Run load	Run	Size (bytes)
Consulair	91	71	162	8	35	9,388
Hippo level 2	37	60	97	4	46	22,366
Manx	24	34	58	3	29	5,052
Megamax	21	83	104	9	26	5,594
Softworks	169	130	199	11	28	29,592

Macptr (Levels of pointer reference in parenthesis)

Compiler	Compile/ assemble	Link	Total	Run load	Run	Size (bytes)
Consulair (40)	88	74	162	10	6	9,494
Hippo level 2 (26)	38	59	97	3	5	22,454
Manx (12)	19	37	56	2	4	5,106
Megamax (40)	27	97	124	10	5	5,684
Softworks (40)	170	120	290	12	3	29,646

Matrix

Compiler	Compile/ assemble	Link	Total	Run load	Run	Size (bytes)
Consulair			no floating point			
Hippo level 2			no floating point			
Manx	25	42	67	5	19	17,348
Megamax	24	100	124	9	34	17,448
Softworks	194	150	344	13	23	49,576

1. "Runload" is the time until the first *print* statement appears on the screen. "Run" is the time from that point until the last *print* statement appears on the screen. The "Run" time thus lets you compare the time spent executing the core of the program. For the Macptr program, "Run load" is the time until the second *print* statement appears—the time to load the pointer array is included here, rather than in the "Run" time. "Size" is a decimal number in bytes. It represents the total size of the program when loaded into RAM and includes the size of uninitialized global and static data, which is not present in the executable on disk.

Table 2.

have had at least five C compilers hit the market (as well as implementations of Pascal, Modula-2, and an improved BASIC). Released from the torment of waiting, many of us will surge lemming-like into our nearest software shop and buy the first C compiler we see. I know I did, anyway!.

The good news is that if it's one of the five I've reviewed here, you will have a sound piece of software that works as advertised. Of course there's always the possibility that there's something radically wrong with one or more of these products that our benchmarks didn't turn up, but I doubt it.

But, as the sane, conscientious, organized, and superbly balanced sort of human being who chooses programming as a profession or hobby, you will probably want to think a bit more carefully about your choice. You might want to consider three areas:

■ What sort of interface do you like? Are you a traditionalist, who likes mneumonics cryptic and unintelligible to those without special knowledge? Do you think icons are best left in Eastern churches? Then the Manx compiler might be for you (it's got great benchmarks too).

Maybe, on the other hand, you think the Mac interface is the greatest thing since sliced toast and the computer for the rest of them is also the computer for us originals. If so, take a close look at Consulair, Megamax, and Softworks compilers.

Or maybe you're somewhere in between. Check out Hippo-C, where you can be Macish one day and UNIXish the next. (Don't forget, they all support the Mac interface in your application. What we're talking about here is just the developer environment.)

■ Do you put a premium on following the

Compiling the Sieve of Eratosthenes benchmark[1] (sec)

Consulair Mac C (from the Finder)	
Load compiler from Finder	16
Compile and assemble	70
Transfer to linker	15
Link	32
Return to Finder	19
Hippopotamus Hippo-C level 2 (from the Hippo operating system)	
Compile pass 1	17
Compile pass 2	9
Assemble	16
Link	60
Manx Aztec C (from the Shell)	
Compile and assemble	20
Link	29
Megamax C (from the Finder)	
Load from Finder	15
Compile to object module (no assemble)	7
Load linker	9
Link	69
Return to Finder	15
Softworks C (from the Finder)	
Load from Finder	25
Compile and assemble	109
Return to Finder	15
Copy .REL file to second disk	25
Load linker	17
Link	72
Return to Finder file to second disk	18
Load linker	17

1. For each compiler tested, this chart specifies each step in compiling, assembling, and linking the Sieve benchmark and returning to a point where the resulting application could be run. Steps which are user dependent (for example, selecting a file name or typing in a command line) are omitted. The user is reminded that steps involving the Finder can vary significantly (perhaps 5 sec either way) depending on the complexity of the desktop and other factors.

Table 3.

"This is a beautifully documented, incredibly comprehensive set of C Function Libraries."
— Dr. Dobb's Journal

COMPLETE SOURCES

■ **PACK 1: Building Blocks I** $149
250 Functions: DOS, Printer, Video, Asynch

■ **PACK 2: Database** $399
100 Functions: B-Trees, Variable Records

■ **PACK 3: Communications** $149
135 Functions: Smartmodem™, Xon/Xoff, Modem-7, X-Modem

■ **PACK 4: Building Blocks II** $149
100 Functions: Dates, Text Windows, Pull-down Menus Data Compression

■ **PACK 5: Mathematics I** $99
35 Functions: Log, Trig, Square Root

■ **PACK 6: Utilities I** $99
Archive, Diff, Replace, Scan, Wipe (Executable Files only)

Lattice™, Microsoft™, DeSmet™, CI-86™ Compilers on IBM PC/XT/AT™
Small and Large Memory Models.
Credit cards accepted
($7.00 handling/Mass. add 5%)

SOFTWARE HORIZONS INC.

165 Bedford Street
Burlington, Mass. 01803
(617) 273-4711

NOVUM ORGANUM

CIRCLE 25 ON READER SERVICE CARD

Manufacturer	Product
Consulair Corp. 140 Campo Dr. Portola Valley, Calif. 94025 (415) 851-3849	Mac C v. 1.0
Hippopotamus Software 1250 Oakmead Pkwy. Ste. 210 Sunnyvale, Calif. 94086 (408) 730-2601	Hippo-C level 2 v. 1.0
Manx Software Systems P.O. Box 51 Shrewsbury, N.J. 07701 (800) 692-1700	Aztec C 68000 level c
Megamax Inc. P.O. Box 851521 Richardson, Texas 75085-1521 (214) 987-4931	Megamax C v. 1.2
Softworks Inc. 607 West Wellington Chicago, Ill. 60657 (312) 975-4030	Softworks C v. 1.0

guidelines the hardware manufacturer has set up? If so, Consulair and Softworks compilers, which are integrated with Apple's own development system, should get special attention from you.

■ If it's only speed and size you care about, check out the benchmarks. But read them with caution. Man does not live on numbers alone.

My list could go on: UNIX freaks should look at Manx and Hippo compilers; beginners might like Hippo-C level 1 (full tutorial on line, and it goes from compile through link with one menu command); Whitesmiths' compiler junkies should remember that it's the starting point for the Softworks compiler; Megamax's has the best benchmarks for a Finder-based system; etc., etc. In the end it comes down to your application. I hope we've given you enough information to begin thinking about your choice.

If not, do what I did. Run out to your nearest software store and buy the first one you see! ■

Michael Rothman is manager of tools programming for Spinnaker Software, Cambridge, Mass. He has held a number of positions in tools programming for mini and microcomputers.

ANNOUNCING . . . COMPUTER LANGUAGE'S

C Seminar/Workshop

Cambridge, MASS
September 1985

Plans are being set now for COMPUTER LANGUAGE's C Seminar. The 2½ day event will be held in the fall of 1985 in beautiful Cambridge.

Details concerning the topics, speakers and dates will be announced soon. This seminar will be the most comprehensive and practical session ever held about the C programming language.

Become involved in this event from the start by filling out this coupon today:

☐ Please send me more information regarding the seminar and Early Bird Discounts.

☐ I would like to present a paper about: ____________________

☐ I suggest the following speakers: ____________________

☐ I would like to see these topics covered: ____________________

Name ____________________
Company ____________________
Address ____________________
City, State, Zip ____________________
Phone ____________________

BA85

Send to: COMPUTER LANGUAGE Seminar
131 Townsend Street • San Francisco, CA 94107

CIRCLE 98 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

BYSO™ LISP

has features that will delight both beginners and advanced programmers. A fast, reliable and complete interpreter for the IBM PC and true compatibles. $125 includes 100 pg. ref. manual and application notes that put you months ahead on useful projects *(making a hybrid language with C, accessing system functions and I/O ports, building your own dialect, etc.).*

LEVIEN INSTRUMENT CO.
P.O. Box 31G
McDowell, VA 24458
703-396-3345

IBM PC is a trademark of the IBM Corp.

A general purpose programming language for string and list processing and all forms of non-numerical computation.

SNOBOL4+ —the entire SNOBOL4 language with its superb pattern-matching facilities • Strings over 32,000 bytes in length • Integer and floating point using 8087 or supplied emulator • ASCII, binary, sequential, and random-access I/O • Assembly Language interface • Compile new code during program execution • Create SAVE files • Program and data space up to 300K bytes RAM

MUSIC ANALYSIS • TEXT PROCESSING • SYMBOLIC MATHEMATICS • ARTIFICIAL INTELLIGENCE

Have you tried SNOBOL4+?

COMPILER PROTOTYPING • CRYPTOGRAPHY • LINGUISTICS • LIST PROCESSING • GAMES •

With **ELIZA** & over 100 sample programs and functions

For all 8086/88 PC/MS-DOS or CP/M-86 systems, 128K minimum 5¼" DSDD, specify DOS/CPM format

Send check, VISA, M/C to: **$95** plus $3 s/h

Catspaw, Inc.
P.O. Box 1123 • Salida, CO 81201 • 303/539-3884

CIRCLE 9 ON READER SERVICE CARD

C Users' Group

Over 40 volumes of public domain software including:

- compilers
- editors
- text formatters
- communications packages
- many UNIX-like tools

Write or call for more details

The C Users' Group
415 E. Euclid • Box 97
McPherson, KS 67460
(316) 241-1065

CIRCLE 5 ON READER SERVICE CARD

PC BASIC PROGRAMMER'S UTILITIES
TRACE—COMPILE—COMPARE

AUTOTRACE powerful single-step or continuous trace in COMPILED or interpreter BASIC. BREAK on variables or line numbers. Change values at any time. RECALL screen displays—save text that scrolls off your screen. Full 80-column printout saves paper; only CHANGED variable values printed. Save trace to disk. This utility finds your logic errors fast!!

AUTOCPL precompiler system. Automates compile/link process. No more manual editing to remove or change code for compilation. No more separate versions to save. Include SLASH/N to remove line numbers. Now compiling is easy, fast, automatic!!

AUTOCOMPARE fast compare of two programs or ASCII files. Prints 132-column or 80-column record of all differences, or save to disk. Includes utility to compare WORDSTAR (R) files. NEW! will unformat indented ASCII Files to allow easy editing with WORDSTAR. Save hours of manual reformatting.

$49.95 each (each includes RAMdisk & SPOOLER)
All 3 for $99.95; S/H $2.50 ea.

TIMESHARE ASSOCIATES, INC. Dept. L
10202 Robinson
Overland Park, KS 66212
(913) 642-7564

CIRCLE 80 ON READER SERVICE CARD

OPT-TECH SORT™

SORT/MERGE program for IBM-PC, XT & AT

Now also sorts dBASE II files!

- 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

Requires DOS, 64K and One Disk Drive

CIRCLE 62 ON READER SERVICE CARD

FoxBASE™
Interpreter/Compiler

- **dBASE II® source compatible**
- Runs 3-20 times faster than dBASE II
- 8087 coprocessor support
- 14 digit precision
- Up to 48 fields per record
- Full type-ahead capabilities
- Provides compact object code and program security
- Twice as many memory variables as dBASE II

FOX SOFTWARE INC.
13330 Bishop Road, P.O. Box 269
Bowling Green, OH 43402
419-354-3981

CIRCLE 29 ON READER SERVICE CARD

YOU CAN C

VIEW v3
The ultimate CP/M disk utility.

- **Edits** any disk sector
- **Displays** allocation bit map
- **Rebuilds** files automatically from selected blocks

If you ever had to recover a crashed disk, you need **VIEW**. Use **VIEW** to patch damaged sectors, un-erase files, examine other disk formats, more

COMPLETE C SOURCE CODE **$59**

BD Software's C Compiler
The most popular CP/M C Compiler comes with **symbolic debugger** and special Western Wares' **ISIS-II** function library **$140**

C-PACK
A disk full of useful CP/M utilities written in C, PEPE, DEL, BACKUP, DPATCH, ECHO, CHX, PAUSE, more Source incl **$19**

Other C Products
EZZAP, ICX, MEDIT, more
Send for a catalog today

CP/M®, Digital Research
ISIS-II ®, Intel Corp

Western Wares 303-327-4898
Box C • Norwood, CO 81423

CIRCLE 41 ON READER SERVICE CARD

Scroll & Recall™
Screen and Keyboard Enhancement for the IBM-PC XT AT & Compatibles

Allows you to conveniently scroll back through data that has gone off the top of your display screen. Up to 27 pages of data can be recalled or written to a disk file.

Allows you to recall, edit and re-enter your previously entered DOS commands and data lines, without retyping.

Very easy to use, fully documented. Compatible with all versions of DOS, monochrome & graphic displays.

$69 - Visa, M/C, Check, COD, POs

Make Your Work Easier!

To Order or to Receive Additional Information, Write or Call:
Opt-Tech Data Processing
P.O. Box 2167 • Humble, Texas 77347
(713) 454-7428
Dealer Inquires Welcome

CIRCLE 64 ON READER SERVICE CARD

ADVERTISER INDEX

	PAGE NO.	CIRCLE NO.
Automata Design Associates	39	1
Amber Systems	25	2
Atlantis Publishing Corp.	37	3
BD Software	60	4
Bendorf & Associates	95	6
Blaise Computing Inc.	88	8
Borland International	cover IV	10
C Systems	78	16
C Users Group	95	5
C Ware	45	11
Catspaw Inc.	95	9
Code Works (The)	54	42
CompuServe	69	7
Computer Helper Industries	77	12
Creative Solutions	38	14
DWB Associates	81	21
Data Base Decisions	87	30
Datalight	39	19
David Smith Software	48	36
Echelon Inc.	42	22
Ecosoft	73	17
Edward Ream	96	–
Entelekon	14	24
Essential Software Inc.	71	18
FairCom	48	26
Fox Software	95	29
Gimpel Software	76	38
Greenleaf Software	58	44
HSC Inc.	49	31
Harvard Softworks	73	47
Hippopotamus Software	39	39
Information Systems Inc.	61	20
Inmos Corp.	8	40
Introl Corp.	59	32
KC Systems	54	45
Laboratory Microsystems Inc.	73	35
Lattice Inc.	74	48
Levien Instrument Co.	95	–
Lifeboat	56	85
Lifeboat	81	87
Lugaru Software Ltd.	64	51
mbp Software & Systems Technology	33	53
MEF Enviromental	86	15
MEF Enviromental	39	55
Manx Software Systems	cover III	69
Megamax Inc	77	13
Micro Focus	18	84
Micro Methods	77	52
Micro Software Developers	50	54
MicroTec Research Inc.	90	49
Micromotion	64	56
Mindbank Inc.	92	63
Nantucket	2	59
Next Generation Systems	52	61
Northwest Computer Algorithms	77	46
Op-Tech Data Processing	95	62
Op-Tech Data Processing	95	64
Phoenix Computer Products Corp.	6, 7	66
Polytron	34	88
Poor Person Software	79	67
ProCode	80	60
Programmer's Connection	89	27
Programmer's Shop (The)	84	70
RR Software	cover II	58
RTL Programming	82	71
Rational Systems Inc.	75	72
Realia Inc.	85	76
SLR Systems	61	73
STSC	4	74
Seven Seas	50	28
Shaw American Technologies	79	34
Soft Craft Inc.	11	75
Softaid Inc.	82	78
Software Horizons	93	25
Solution Systems	67	94
Solution Systems	23	37
Solution Systems	84	91
Solution Systems	84	92
Spruce Technology Corp.	24	33
Systems Guild	12	89
Thunder Software	86	65
Timeshare Associates Inc.	95	80
UniPress	79	81
Uniworks	1	82
Western Wares	95	41
Whitesmiths Ltd.	40	43
Wizard Systems	82	86
Workman & Associates	82	68
Zeducomp	46	83

The index on this page is provided as a service to our readers. The publisher does not assume any liability for errors or omissions.

C Source Code

RED

Full Screen Text Editor

IBM PC, Kaypro, CP/M 80 and CP/M 68K systems.

- RED is fast! RED uses all of your terminal's special functions for best screen response. RED handles files as large as your disk automatically and quickly.
- RED is easy to use for writers or programmers. RED's commands are in plain English.
- RED comes with complete source code in standard C. RED has been ported to mainframes, minis and micros.
- RED comes with a Reference Card and a Reference Manual that provides everything you need to use RED immediately.
- RED is unconditionally guaranteed. If for any reason you are not satisfied with RED your money will be refunded promptly.

RED: $95

Manual: $10

edward k ream

Call or write today for for more information:
Edward K. Ream
1850 Summit Avenue
Madison, WI 53705
(608) 231-2952

To order:

Either the BDS C compiler or the Aztec CII compiler is required for CP/M80 systems. Digital Research C compiler v1.1 is required for CP/M 68K systems. No compiler is required for IBM or Kaypro systems.

Specify both the machine desired (IBM, Kaypro or CP/M) and the disk format described (8 inch CP/M single density or exact type of 5¼ inch disk).

Send a check or money order for $95 ($105 U.S. for foreign orders). Sorry, I do NOT accept phone, credit card, or COD orders. Please do not send purchase orders unless a check is included. Your order will be mailed to you within one week.

Dealer inquiries invited.

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 ______________

April issue. Not good if mailed after August 31, 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.

2/4

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 ______________

April issue. Not good if mailed after August 31, 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.

2/4

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.

BIA5

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.

BIA5

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

They said it couldn't be done. Borland Did It. Turbo Pascal 3.0

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!

	TURBO 3.0	TURBO 2.0	MS PASCAL
COMPILATION SPEED	8 sec.	16 sec.	206 sec
EXECUTION SPEED	9 sec.	13 sec.	20 sec.
CODE SIZE	12 K	12 K	35 K
BUILT-IN INTERACTIVE EDITOR	YES	YES	NO
ONE STEP COMPILE (NO LINKING NECESSARY)	YES	YES	NO
COMPILER SIZE	35K	35K	300K+
TURTLE GRAPHICS	YES	NO	NO
BCD OPTION	YES	NO	NO
PRICE	$\$69^{95}$	$\$54^{95}$	$\$295^{00}$

(*) **Benchmark** run on an IBM PC using MS Pascal version 3.2 and the DOS linker version 2.6. The 179 line program used is the "Gauss-Seidel" program out of Alan R. Miller's book: *Pascal programs for scientists and engineers* (Sybex, page 128) with a 3 dimensional non-singular matrix and a relaxation coefficient of 1.0.

The best just got better: Introducing Turbo Pascal 3.0

We just added a whole range of exciting new features to Turbo Pascal:

- First, the world's fastest Pascal compiler just got faster. Turbo Pascal 3.0 compiles twice as fast as Turbo Pascal 2.0! No kidding.
- Then, we totally rewrote the file I/O system, and we also now support I/O redirection.
- For the IBM PC versions, we've even added "turtle graphics" and full tree directory support.
- For all 16 Bit versions, we now offer two additional options: 8087 math coprocessor support for intensive calculations and Binary Coded Decimals (BCD) for business applications.
- And much much more.

The Critics' Choice.

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 version of Turbo Pascal will soon be announced, and before the end of the year, Turbo Pascal will be running on most 68000 based microcomputers.

An Offer You Can't Refuse

Until June 1st, 1985, you can get Turbo Pascal 3.0 for only $69.95. Turbo Pascal 3.0, equipped with either the BCD or 8087 options, is available for an additional $39.95 or Turbo Pascal 3.0 with both options for only $124.95. As a matter of fact, if you own a 16 Bit computer and are serious about programming, you might as well get both options right away and save almost $25.

Update policy

As always, our first commitment is to our customers. You built Borland and we will always honor your support.

So, to make your upgrade to the exciting new version of Turbo Pascal 3.0 easy, we will accept your original Turbo Pascal *disk* (in a bend-proof container) for a trade-in credit of $39.95 and your Turbo87 original *disk* for $59.95. This trade-in credit may only be applied toward the purchase of Turbo Pascal 3.0 and its additional BCD and 8087 options (trade-in offer is only valid directly through Borland and until June 1st, 1985).

NOT COPY-PROTECTED

TURBO PASCAL

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

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:
☐ 5¼" ☐ 8" ☐ 3½"

For update: original Turbo disk must accompany order

YES! I want the Best!

Please send:		Quantity
Pascal 3.0	$ 69.95	____
Pascal w/8087	$109.90	____
Pascal w/BCD	$109.90	____
Pascal w/8087 & BCD	$124.95 (SAVE $24.90)	____

• These prices include shipping to all U.S. cities. All foreign orders add $10 per product ordered.

Trade-in Credit Claimed: ____

Amount: (CA 6% tax) ____

Payment: VISA MC BankDraft Check

Credit Card Expir. Date: ____/____

Name on Card: ____

Card #: ____

Name: ____

Shipping Address: ____

City: ____ Zip: ____

State: ____

Telephone: ____

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.

20

BORLAND INTERNATIONAL

Software's Newest Direction
4113 Scotts Valley Drive
Scotts Valley, California 95066
TELEX 172373

Turbo Pascal is a registered trademark of Borland International, Inc.

CIRCLE 10 ON READER SERVICE CARD