

COMPUTER LANGUAGE

\$2.95
TM

VOLUME 1, NUMBER 4

DECEMBER 1984

FRED: A LANGUAGE
WITHIN FRAMEWORK

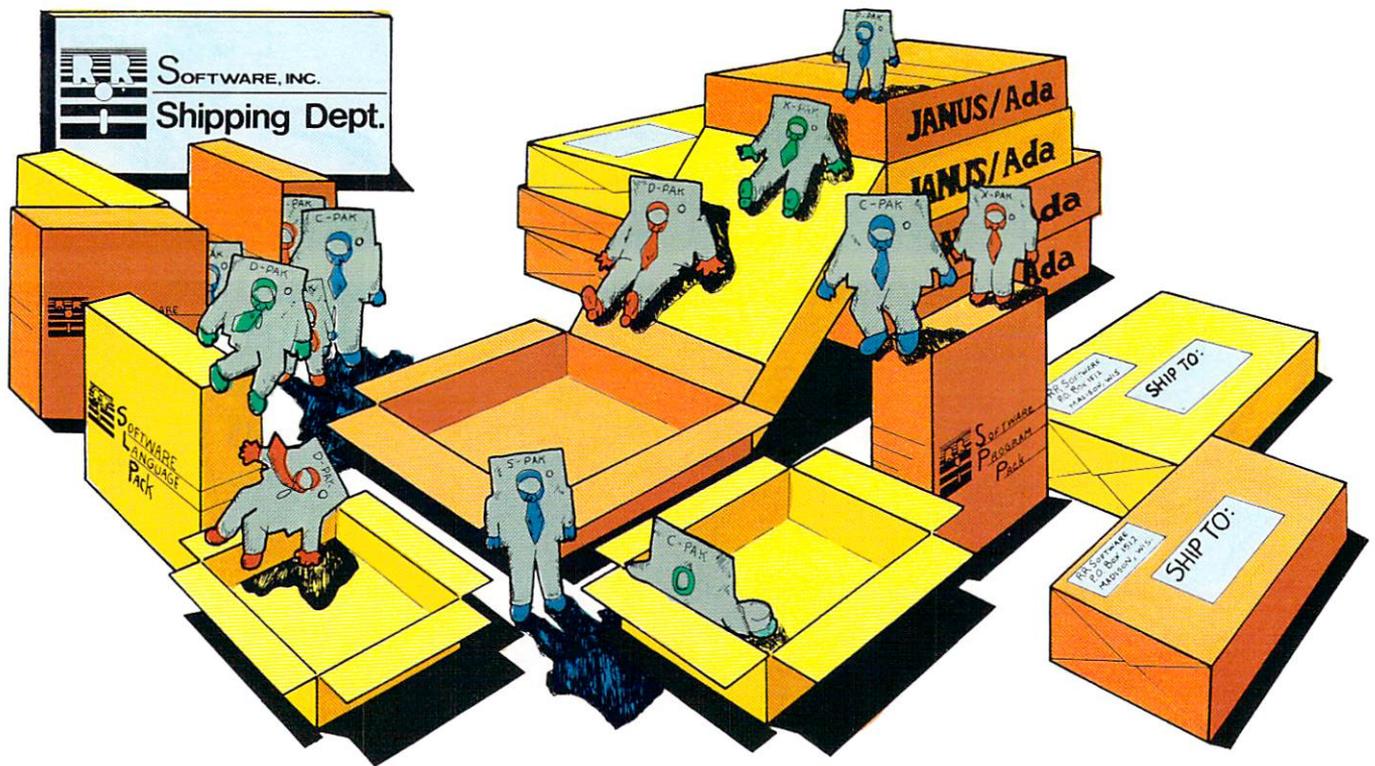
GODBOUT TELLS THE
S-100 STORY

NEW EXOTIC LANGUAGE
CALLED OMNI

SIX PASCAL COMPILERS
COMPARED

EXPLORATORY PROGRAMMING

WE'VE GOT YOUR PACKAGE!!



We offer you the most flexible, cost efficient means of introducing your programming staff to the Ada Language. **You** can choose the level of Support **you** need, when you need it! These Janus/Ada packages are customer-tested and available now...

(C-Pak) Introductory Janus/Ada Compilers
(D-Pak) Intermediate Janus/Ada Systems
(S-Pak) Advanced Janus/Ada Systems
(P-Pak) Janus/Ada Language Translators

Janus/Ada "Site" Licenses
Janus/Ada Source Code Licenses
Janus/Ada Cross Compilers
Janus/Ada Maintenance Agreements

Coming Soon: New Computer and Operating Systems Coverage

Selected Janus/Ada packages are available from the following:

National Distributors

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

Soft-Net
 5177 Richard, Suite 635
 Houston, TX 77056
 (713) 933-1828

AOK Computers
 816 Easley St., Suite 615
 Silver Springs, MD 20910
 (310) 588-8446

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

Compview Products, Inc.
 1955 Pauline Blvd., Suite 200
 Ann Arbor, MI 48103
 (313) 996-1299

International Distributors

Micronix
 11 Blackmore St.
 Windsor 4030
 QLD, Australia
 (07) 57 9152

Progreso
 155, rue du Fauburg
 St. Denis
 75010 Paris
 (1) 205-39-47

Lifeboat of Japan
 S- 13-14, Shiba
 Minato-Ku
 Tokyo 108 Japan
 03-456-4101

CP, M, CP/M-86, CCP, M-86 are trademarks of Digital Research, Inc.
 *ADA is a trademark of the U.S. Department of Defense
 MS-DOS is a trademark of Microsoft

© Copyright 1983 RR Software



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



Flight Simulation



Guidance Systems



ASW Weapons Systems

SOFTWARE ENGINEERING

For The New Defense Technology

Goodyear Aerospace is bringing computer-based systems technology to the new wave of defense programs destined for future deployment. Our current engineering contracts, combined with the new projects we were recently assigned, call for more of the Technical talent responsible for the success of our diverse electronic and systems products. Look into these immediate opportunities.

Software Engineers - a number of challenging opportunities for Software Professionals to work on diverse systems development and applications. BSEE, CS, or related Technical education and minimum 3 years practical advanced computer systems experience including higher level languages required.

Systems Engineers - BSEE, CS, or related field with opportunities for MS & PhD level candidates. Apply advanced hardware and software technology to simulation, guidance, visual systems, weapons, or other specialized projects. Previous

experience with a major DOD contractor, particularly missile systems preferred.

Scientific Programmer Analysts - Analyze complex engineering and mathematical problems in support of advanced systems engineering and development. BSEE or CS and ability to utilize modern computer resources to maximum capability essential.

Digital Design - Using detailed Systems level input, you will design VLSI/VHSIC circuits from the component level. MS/BSEE and 5 years related experience required.

Goodyear Aerospace offers qualified candidates a total compensation package commensurate with your current and projected skills; including generous company benefits, educational assistance, and an attractive environment conducive to large city, suburban or even rural lifestyles. Our stability and continued projected growth provides engineering professionals the opportunity to utilize and expand on their career potential without changing employers. For details or a confidential interview appointment in your area, please send your resume indicating your area of interest to:

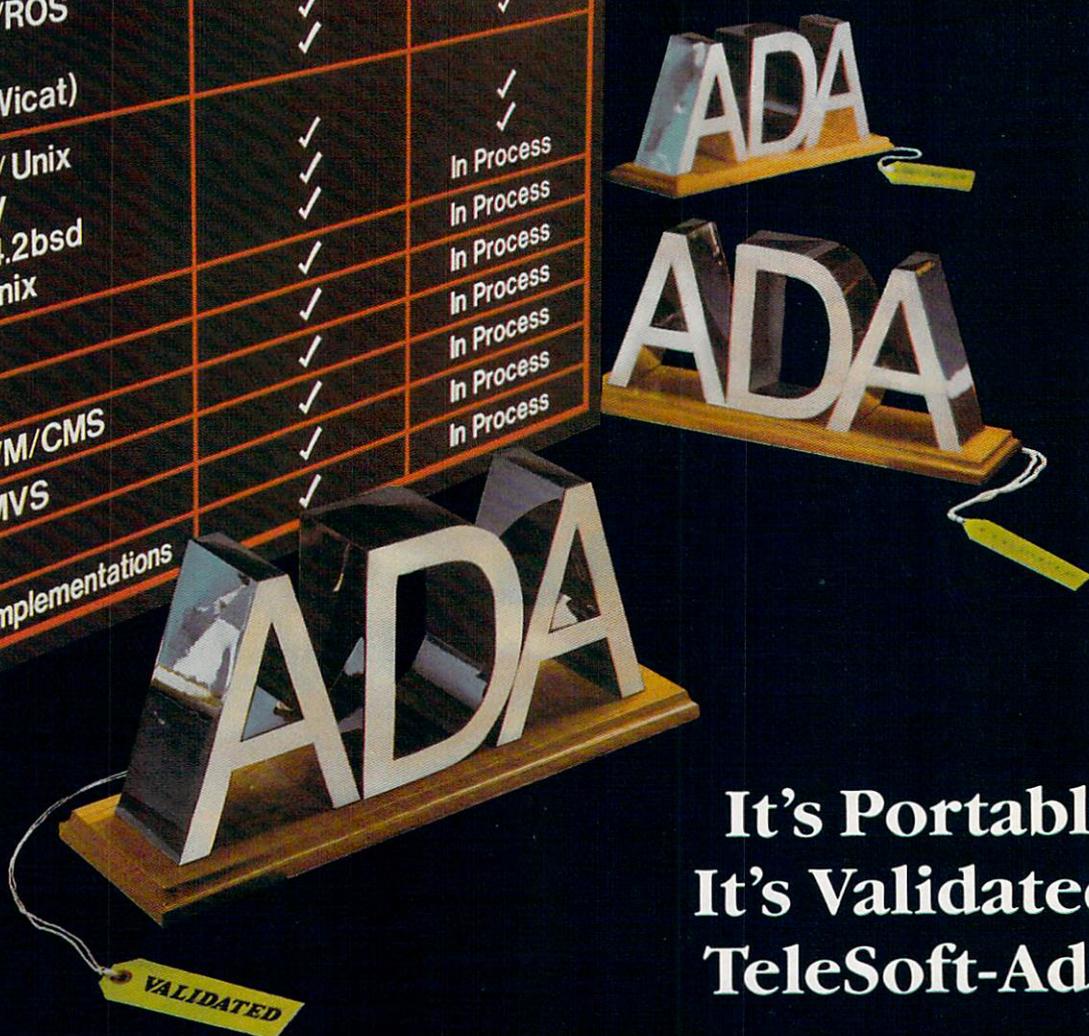
GOODYEAR AEROSPACE

Dept. 131-OH
Akron, Ohio 44315

Equal Opportunity Employer, M/F/H

Ada's First Family

Compiler	In Production Use	Validated
MC68000/ROS TeleSoft Labtek (Wicat)	✓	✓
MC68000/Unix System V	✓	In Process
Berkley 4.2bsd	✓	In Process
Unisoft Unix	✓	In Process
VAX/VMS	✓	In Process
VAX/Unix	✓	In Process
IBM 370/VM/CMS	✓	In Process
IBM 370/MVS	✓	In Process
IBM PC	✓	In Process
Additional Implementations		



**It's Portable.
It's Validated.
TeleSoft-Ada.**

A growing company in San Diego has taken a giant step in bringing Ada to the software development world.

TeleSoft, the company that fielded the first Ada compiler and leads the market in user experience with over 450 installed, has validated the first in its family of portable Ada compilers.

TeleSoft's Ada compiler family is portable across most of today's popular host systems and development environments: MC68000/Unix, MC68000/ROS, Digital's VAX/VMS and VAX/Unix, IBM 370 VM/CMS and MVS. And development is well underway on Ada compilers for nearly a dozen other systems.

These compilers are all that Ada was intended to be: Truly portable, with complete support environments, and a clear growth path to exceptional compilation and execution speeds. TeleSoft has done this by developing technology

which minimizes machine-dependent software and allows fast adaptation to new environments.

TeleSoft's announcement is the first big step towards a new era in the software industry—based on components, true software portability, and Ada standardization.

TeleSoft™

10639 Roselle Street · San Diego, Ca. 92121 · (619) 457-2700

Ada is a trademark of the Department of Defense; VAX and VMS are trademarks of Digital Equipment Corporation; UNIX is a trademark of Bell Laboratories; IBM 370, IBM PC, CMS and MVS are trademarks of IBM Corporation; MC68000 is a trademark of Motorola Corporation. TeleSoft-Ada is a trademark of TeleSoft. Copyright 1983, TeleSoft.

CIRCLE 74 ON READER SERVICE CARD

COMPUTER LANGUAGE

ARTICLES

Exploratory Programming

by Michael Ham

Coding style is greatly influenced by the syntax and tools provided by the available programming language. As an extensible language, Forth promotes a kind of "revise as you go" coding style that can help programmers discover how to progressively improve the speed and efficiency of their programs.

27

Fred—More than just a macro facility within Framework

by Darryl Rubin

Within Framework, a new integrated software package from Ashton-Tate, exists a programming language named Fred that has many of the powerful features of LISP, Smalltalk, BASIC, and C. Here we discover how to write and run Fred programs and how to develop some useful, undocumented Fred routines.

33

Enhancing Source Code Control under UNIX, Part II

by Luke C. Dion and Alan Filipski

Last month we described how a front-end interface to the standard UNIX SCCS utility set could solve the problem posed when many people are trying to store and access their files simultaneously. In this issue, Dion and Filipski discuss the tools necessary to design and implement such an interface.

41

What Day is It, Exactly?

by Joe Celko

The format for writing dates on a computer is by no means standard. Using Julianized dates, this author provides the various algorithms necessary to establish an accurate, consistent calendar program based on the U.S. military's format for calendar dates.

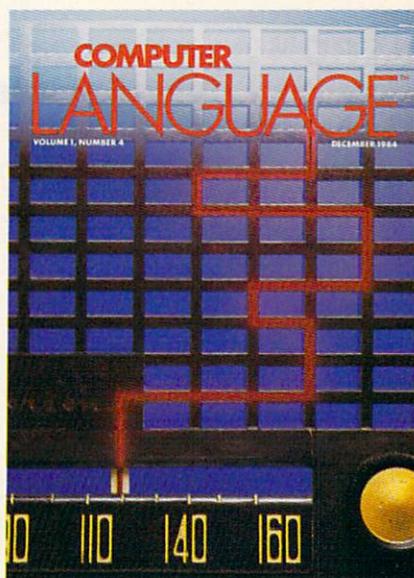
47

Customize your High-Level Language

by Charles K. Ballinger

Assembly subroutines are often designed to interface with high-level programs in order to execute a specific task when called. This author shows how easily such an interface can be achieved by describing how he married Microsoft's COBOL-80 language with assembler to control his printer.

51



DEPARTMENTS

Editor's Notes _____	5
Feedback _____	9
Back to the Drawing Board _____	15
Designers Debate _____	21
The art of hacking	
Public Domain Software Review _____	55
Exotic Language of the Month Club _____	58
OMNI: One person's language	
ComputerVisions _____	63
Bill Godbout, chairman and CEO of CompuPro	
Software Reviews _____	69
A comparison of six Pascal compilers	
Advertiser Index _____	80

WHY DEBUG YOUR PROGRAM IN ASSEMBLY LANGUAGE WHEN YOU WROTE IT IN ONE OF THESE...

ATRON Announces Source Level Software Debugging

Without source level debugging, the programmer must spend time mentally making translations between assembly language and the C, PASCAL, or FORTRAN source code in which the program was written. These tedious translations burn up valuable time which should be spent making critical product schedules. The low level hex and symbolic debuggers available today are superseded by ATRON'S solution — Source Probe.

HOW TO SINGLE STEP YOUR SOURCE CODE AND KEEP CRITICAL DATA IN VIEW

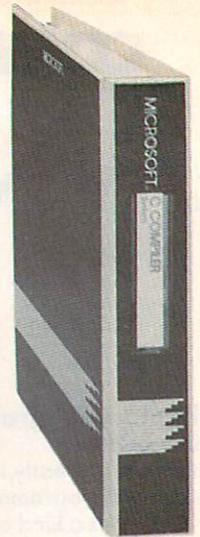
With Source Probe, you can step your program by source code statements. While stepping, a window which you define can display critical high level data structures in your program. The next several source code statements are also displayed to give you a preview of what the program will do

HOW TO DISPLAY DATA IN MEANINGFUL FORMATS

Why look at program data in hex when you defined it to be another data type in your program. Source Probe provides a formatted print statement to make the display of your variables look like something you would recognize. You can specify data symbolically too.

FIND A BUG — FIX IT RIGHT NOW

Source Probe provides an on-line text editor to allow you to log program corrections as you find them while debugging. With on-line display and editing of source files, the time lost printing and looking through program listings can be eliminated.



A SNAP SHOT OF REAL TIME PROGRAM EXECUTION — BY SOURCE CODE !

When Source Probe is running on ATRON'S PC PROBE hardware, the real time execution of the program is saved. You can then view your source code as it executed in real time — including all the changes the program made to your data variables.

HOW TO FIND A BUG WHICH OVERWRITES MEMORY

When running on PC PROBE, the Source Probe can trap a bug which overwrites a memory location. Because complex pointers are normally used in high level language programming, this bug occurs frequently and is very difficult to find.

A BULLET PROOF DEBUGGER

What good is a debugger that can be wiped out by an undebugged program? With Source Probe running on PC PROBE, the software is write protected and cannot be changed.

ATRON PROVIDES THE DEBUGGING TOOLS WHICH FIT YOUR PROBLEM

- PC PROBE — A hardware aid to symbolic software debugging
- SOFTWARE PROBE — A symbolic debugger, runs without PC PROBE
- SOURCE PROBE — A source level debugger, versions run with or without PC PROBE
- PERFORMANCE AND TIMING ANALYZER — For finding where your program spends its time

WE HAVE HUNDREDS OF HAPPY CUSTOMERS

ATRON produced the first symbolic debugger for the PC and the first hardware aided debugging tool — PC PROBE. We have hundreds of happy customers who have made their schedules because of ATRON debugging tools. Why waste more time — call us today!



20665 FOURTH STREET • SARATOGA, CA 95070 • (408) 741-5900

CIRCLE 4 ON READER SERVICE CARD

Editor's Notes

Today a growing number of people are becoming involved in the art and science of programming. Many of the new entrants to our industry are unaware of the time when hobbyists would wire wrap their own printed circuit boards on workbenches in their garages to produce homemade, single-board systems that had a only few kilobytes of memory to play with.

Back in 1975, mainframe programming was for the privileged few who had the clearance badges to those sacred, sterile computer rooms in large corporations and government facilities.

The microcomputer world has taken a few quantum leaps since then, and many people today feel that small systems programming issues have become even more complex than mainframe issues. The technical hurdles that now face programmers challenge intellectual and creative abilities to an extreme.

For example, in the programming world of the 1980s, debates are fierce on whether the C programming language is only good for systems programming jobs and whether "big" languages like Ada and COBOL can ever be successfully implemented in their standard forms on a microcomputer.

COMPUTER LANGUAGE will continue to be a forum for the discussion of issues like these and other subjects that are important to today's programmer.

On that point, this month I'd like to announce our plans to begin publishing theme issues on an planned basis. We are now in the process of designing a 1985 editorial calendar that will map out our future coverage of subjects judged to be of current interest to people in our industry.

Next February we will begin with our first theme issue, which will be devoted to C. We have tentative plans for theme

issues on BASIC, Modula-2, artificial intelligence, COBOL, and exotic languages in the following months.

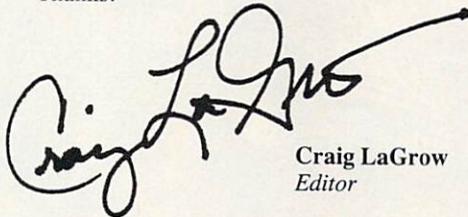
Each theme issue will also include a product wrap-up table that will analyze all the important products in the field being covered (e.g., compilers/interpreters for all language-oriented issues). In the Software Reviews department this month, we begin this product analysis strategy with our review of six Pascal compilers.

Also in the works for next year will be our first East Coast *COMPUTER LANGUAGE* technical seminar on C. Geared primarily toward experienced C programming professionals, this seminar will feature lectures and workshops led by well-known figures in the C world.

We are also considering sponsoring other technical speaking events in major cities around the country. What would you like to see covered in such a seminar if you were to attend? Who do you think would be the speakers most qualified to lead technical workshops?

I encourage you to write in with your ideas on the subjects *you* think we should be considering for our upcoming technical seminar and also for future theme issues. Tell us how we might also improve the criteria by which products are compared in our Pascal wrap-up table.

And, in general, now that you've had a chance to see how we've covered programming subjects in the past three issues, I hope you'll continue to write in with your thoughts on how we can better improve our editorial coverage, style, or anything else you might have in mind. Thanks!



Craig LaGrow
Editor

COMPUTER LANGUAGE

EDITOR

Craig LaGrow

MANAGING EDITOR

Regina Starr Ridley

TECHNICAL EDITOR

John Halamka

EDITORIAL ASSISTANT

Hugh Byrne, Lorilee Biernacki

CONTRIBUTING EDITORS

Burton Bhavisyat, Steve Heller,
Tim Parker, Ken Takara

INDUSTRY NEWS CONSULTANT

Bruce Lynch

ADVERTISING SALES

Jan Dente

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 BBS: (415) 957-9370, 7 bit, even parity, 300/1200 baud. CompuServe account: "GO CLM".

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: \$24.00 per year (U.S.); \$30.00 per year (Canada and Mexico). Subscription prices for outside the U.S., Canada, and Mexico: \$36.00 (surface mail), \$54.00 (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.

Reprints: Copyright 1984 by COMPUTER LANGUAGE Publishing Ltd. All rights reserved. Reproduction of material appearing in COMPUTER LANGUAGE is forbidden without written permission.

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

COMPUTER LANGUAGE is a registered trademark owned by the magazine's parent company, CL Publications. All material published in COMPUTER LANGUAGE is copyrighted © 1984 by CL Publications, Inc. All rights reserved.

NEW from BORLAND!

TURBO TOOLBOX & TURBO TUTOR

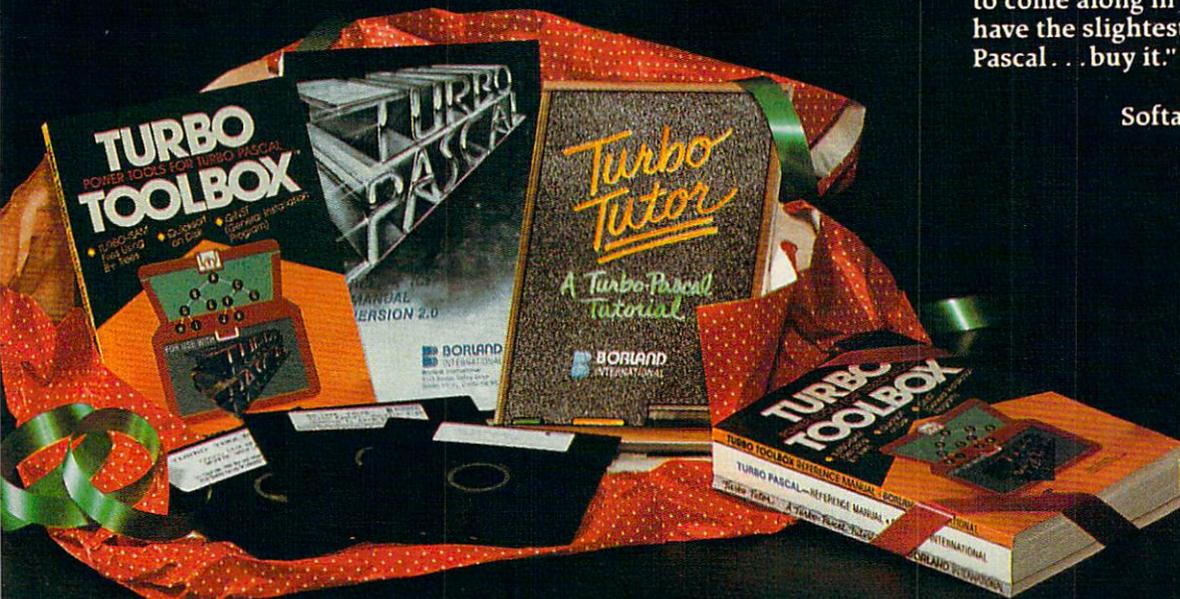
"TURBO is much better than the Pascal IBM sells."

Jerry Pournelle,
Byte, July 1984

"TURBO PASCAL appears to violate the laws of thermodynamics.

You won't find a comparable price/performance package anywhere. It is simply put, the best software deal to come along in a long time. If you have the slightest interest in Pascal... buy it."

Bruce Webster,
Softalk IBM: March 1984



BORLAND INTERNATIONAL GIFT PACK

ONLY **\$99.95**
A SAVINGS OF \$30!

What a gift for you and your friends! The extraordinary TURBO PASCAL compiler, together with the exciting new TURBO TOOLBOX and new TURBO TUTOR. All 3 manuals with disks for \$99.95.

TURBO PASCAL Version 2.0 (reg. \$49.95). The now classic program development environment still includes the FREE MICROCALC SPREAD SHEET. Commented source code on disk

- Optional 8087 support available for a small additional charge

NEW! TURBO TOOLBOX (reg. \$49.95). A set of three fundamental utilities that work in conjunction with TURBO PASCAL. Includes:

- TURBO-ISAM FILES USING B+ TREES. Commented source code on disk
- QUIKSORT ON DISK. Commented source code on disk
- GINST (General Installation Program)

Provides those programs written in TURBO PASCAL with a terminal installation module just like TURBO'S!

- NOW INCLUDES FREE SAMPLE DATABASE... right on the disk! Just compile it, and it's ready to go to work for you. It's a great example of how to use TURBO TOOLBOX and, at the same time, it's a working piece of software you can use right away!

NEW! TURBO TUTOR (reg. \$29.95). Teaches step by step how to use the TURBO PASCAL development environment—an ideal introduction for basic programmers. Commented source code for all program examples on disk.

30 DAY MONEY BACK GUARANTEE These offers good through Feb. 1, 1985

For VISA and MASTERCARD order call toll free: **1-(800)-255-8008 1-(800)-742-1133**
(Lines open 24 hrs., 7 days a week) Dealer and Distributor inquiries welcome (408) 438-8400

CHOOSE ONE (please add \$5.00 for handling and shipping U.S. orders)

<input type="checkbox"/> All Three-Gift Pack	\$ 99.95 + 5.00 SPECIAL!	<input type="checkbox"/> Turbo Toolbox	\$49.95 + 5.00
<input type="checkbox"/> All Three & 8087	139.95 + 5.00 SPECIAL!	<input type="checkbox"/> Turbo Tutor	29.95 + 5.00
<input type="checkbox"/> Turbo Pascal 2.0	49.95 + 5.00	<input type="checkbox"/> Turbo 8087	89.95 + 5.00

Check _____ Money Order _____ VISA _____ MasterCard _____

Card #: _____ Exp. date: _____ Shipped UPS

My system is: 8 bit _____ 16 bit _____

Operating System: CP/M 80 _____ CP/M 86 _____ MS DOS _____ PC DOS _____

Computer: _____ Disk Format: _____

Please be sure model number & format are correct.

NAME: _____

ADDRESS: _____

CITY/STATE/ZIP: _____

TELEPHONE: _____

California residents add 6% sales tax. Outside U.S.A. add \$15.00 (if outside of U.S.A. payment must be by bank draft payable in the U.S. and in U.S. dollars). Sorry, no C.O.D. or Purchase Orders.

H20

 **BORLAND**
INTERNATIONAL

4113 Scotts Valley Drive
Scotts Valley, California 95066
TELEX: 172373

CIRCLE 6 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.

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.

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.

Target Microprocessors

8086/80186
8096
8080/8085
8051
8048
Z8002
Z80
Z8
NSC 800
9800 ...

68000/08/10
6809
6800/01/02
6805
6301
6305
6500
G65SCXX, G65SC1XX
R65C00, R6500/1...
others

Host Computers

DEC VAX, PDP-11
DG MV-Series
DG Eclipse
Apollo
UNIX Systems
IBM PC
Data General/ONE
HP 150
DEC Rainbow
others

Software Tools

C Compilers
Pascal Compilers
Assemblers
Linking Loaders
Librarians
Conversion/
Download
Communications
VT-100 Emulator

for Serious Software Developers

MICROTEC is a trademark of Microtec Research, Inc. Santa Clara, CA

**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

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.

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**

C Pascal Assemblers Simulators

FEEDBACK

NSWEEP defended

Dear Editor:

I disagree with Tim Parker's comment about front-end programs for CP/M 80.

He apparently hasn't seen NSWEEP (NewSWEEP), a 12K bundle of dynamite that not only replaces REN, TYPE, PIP, and ERA, but also has such features as squeeze and unsqueeze, sets file status, and more.

Let's see how fast brute force can move half a dozen files to another disk if the names are unrelated so wild cards can't do it with PIP, then erase all of them. Try squeezing three of the six at the same time while unsqueezing the other three! With NSWEEP205 you hit "t" to tag the files you want to move as you space through the alphabetical list of files, "Q" to indicate the squeeze routine, "R" to indicate "reverse", (squeeze the unsqueezed files and unsqueeze the squeezed ones), a drive and user specification for the destination, and NSWEEP does the work. When it's done, hit "a" for retag, "e" to erase, "t" to indicate the tagged files are the ones to erase, and "n" to indicate no prompting before erasing.

Now I'll grant that you could write a submit file to do it, or with ZCPR or some other program you could use multiple commands on a single line, but you'll spend a lot more time doing it and if you misspell one of the names of the files you'll be done in. NSWEEP lets you view or print out squeezed files without having a separate program for that function or unsqueezing the files first.

SWEEP version 4.0 ran about 28K and had fewer features than NSWEEP at 12K. DISK and WASH are primitive by comparison.

By the way, NSWEEP told me how many K bytes the programs to be copied were (cumulatively) and a simple mash on two keys would tell me whether there was space enough on the destination drive, though I wouldn't have the size of files before/after squeezing if I were doing that at the same time. If true programmers eschew these easy features, it's no wonder so many programs are lousy.

Benjamin H. Cohen
Chicago, Ill.

Ooops

Editor's Note: To err is human, right? In our November issue we published a letter from Gary Nemeth of Cleveland, Ohio, that was supposed to include a Forth version of the Quicksort algorithm. Instead, we re-published the Fortran version of an issue earlier. Our humble apologies.

Will the real Forth Quicksort please stand up? Presented in Listing 1 is Nemeth's Forth Quicksort, which was converted to IBM PC display management. In this version, the data-dependent words COMPARE and EXCHANGE are factored. The first line is the machine-dependent part and thus must be ported (debugged first). The rest of the program will run on any Forth computer.

Promotional cover

Dear Editor:

I was getting a little worried about your monthly publication after three months had gone by. Recently, however, I have begun to receive *COMPUTER LANGUAGE*.

I was a little upset at first because the first issue I received was the premier issue (the one I had purchased at the book store). But I reread it and passed it on to a friend. Soon my second issue came. And then there were two, and they were both good.

```
( ECHO RCV>PAD RCVBLOCKS ) ( 8/03/84 GSN)
NEED PRINT2 ( 5/31/84 GSN)
: FRAME BEGIN AKEY ASCII * = UNTIL ;
  VARIABLE PAD 1K ALLOT
: RCV>PAD FRAME 1K 0 DO AKEY PAD I + C! LOOP ;
: .PAD 16 0 DO CR I 3 .R SPACE PAD I 64 * + 64 TYPE LOOP ;
: STASH PAD SWAP BLOCK 1K CMOVE UPDATE FLUSH ;

: RBLOCK RCV>PAD .PAD DUP STASH LIST ;
: RBLOCKS 1+ SWAP DO I RBLOCK LOOP ;
: XBLOCK PRINT2 ." *" BLOCK 1K TYPE ME ;
: XBLOCKS 1+ SWAP DO I XBLOCK 4 DELAYS LOOP ;

: =BLOCK RCV>PAD DUP . BLOCK 1K PAD -TEXT ABORT" !@# " ;
: =BLOCKS 1+ SWAP DO I =BLOCK LOOP ;
;S

: QUICKSORT ; ( from Forth Dimensions V5#5 7/26/84 GSN)
NEED FORTH79 NEED RECURSE
: K@ 2* CRTSEG LC@ ; : K! 2* CRTSEG LC! ; VARIABLE MIDDLE
: MIDDLE@ ( a a -- ) OVER - 2/ + K@ MIDDLE ! ;
: COMPARE ( a -- ) K@ MIDDLE @ - ;
: EXCHANGE ( a a -- ) 2DUP K@ SWAP K@ ROT K! SWAP K! ;
: SORT ( a low a high -- ) 2DUP > IF 2DROP ELSE
  2DUP 2DUP MIDDLE@ ( -- 1 h 1 h )
  BEGIN SWAP BEGIN DUP COMPARE 0< WHILE 1+ REPEAT
  SWAP BEGIN DUP COMPARE 0> WHILE 1- REPEAT
  2DUP > NOT IF 2DUP EXCHANGE 1 -1 D+ THEN
  2DUP > UNTIL SWAP ROT ( -- start sl+ el- end )
  2OVER 2OVER - ROT ROT - < IF 2SWAP THEN ( smallest first )
  RECURSE RECURSE THEN ;
: SS ( Sort Screen ) 0 1359 SORT ; ALIAS QSS SS
```

Listing 1.



I have one question: on page 8 of both issues appears an ad for *COMPUTER LANGUAGE*. The ad shows a picture of the premier issue which looks a lot like my premier issue, except its cover is a little different (read totally different). I'm not complaining, it's just those articles in the ad interest me a great deal. I was wondering where I could get that issue (or an explanation). Thank you.

Dan Efron
St. Louis Park, Minn.

Editor's Note: Before we began making preparations to publish the first issue of COMPUTER LANGUAGE, we needed to test the waters with a hypothetical promotional issue. At that point the publisher, Carl Landau, and I were not even sure COMPUTER LANGUAGE would ever come to be.

Since that early time we have published four real issues, yet many readers are also interested in those article subjects which appeared on the promotional cover.

So, in the interests of those who have written in so far with this concern, I have consigned for future publication each and every one of the article subjects referred to on the promotional cover.

In the November issue you may have noticed our natural language processing article, and soon to come will be articles on C screen editors and macro libraries.

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 its 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" SSDD 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.

BDS Software

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

CIRCLE 5 ON READER SERVICE CARD

CL for professionals

Dear Editor:

I have almost finished reading the premier issue of *COMPUTER LANGUAGE* and am surprised and extremely delighted with it. This is the only publication I have read that is targeted to the professional software designer.

The article on True BASIC was quite an eye catcher as I have been using a language almost exactly like it for several years. BASIC09 (Microware Systems, Des Moines, Iowa) contains the constructs for structured programming, the ability to create library functions and link to them at run time, and most of the other desirable features in True BASIC. In addition, BASIC09 has an interactive editor to catch the simple mismatched quotes or misused keyword, as well as a compiling debugger to pick up the likes of unterminated structures and other logical errors. BASIC09 produces a compiled "I code" upon exiting the editor and need not be compiled each time the program is run.

Don Loughlin
Hauppauge, N.Y.



More on mainframes

Dear Editor:

Before I say anything negative, I want to say quickly that I think *COMPUTER LANGUAGE* is very successful, and the proof of my high opinion of your work is that I am willing to spend the time to write this letter.

If I have any overall reservations with *COMPUTER LANGUAGE* it is that the premier issue is too oriented toward microcomputers. The old world of mainframes and so on may be duller but it remains and will probably always be the place where most of the computing (and programming) is done.

Your layout is attractive and intelligent; I like it. Your articles are so-so. One bad sign is a tendency toward quick superficial articles. I would say that there is a minimal market for a popular magazine on computer languages and *COMPUTER LANGUAGE* must be at least as technically tough as the *Scientific American* (more would be better).

David Kleinecke
Santa Barbara, Calif.



Eco-C Compiler

Release 3.0

We think Rel. 3.0 of the Eco-C Compiler is the fastest full C available for the Z80 environment. Consider the evidence:

Benchmarks*

(Seconds)

Benchmark	Eco-C	Aztec	Q/C
Seive	29	33	40
Fib	75	125	99
Deref	19	CNC	31
Matmult	42	115	N/A

*Times courtesy of Dr. David Clark
CNC - Could Not Compile
N/A - Does not support floating point

We've also expanded the library (120 functions), the user's manual and compile-time switches (including multiple non-fatal error messages). The price is still \$250.00 and includes Microsoft's MACRO 80. As an option, we will supply Eco-C with the SLR Systems assembler - linker - librarian for \$295.00 (up to six times faster than MACRO 80).

For additional information, call or write:



ECOSOFT INC. (317) 255-6476
6413 N. College Ave. • Indianapolis, Indiana 46220



NEW RELEASE

CIRCLE 22 ON READER SERVICE CARD

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.



Laboratory Microsystems Incorporated
Post Office Box 10430, Marina del Rey, CA 90295
Phone credit card orders to (213) 306-7412



CIRCLE 35 ON READER SERVICE CARD

FOR TRS-80 MODELS 1, 3 & 4
IBM PC, XT, AND COMPAQ

The MMSFORTH System. Compare.

- The speed, compactness and extensibility of the MMSFORTH total software environment, optimized for the popular IBM PC and TRS-80 Models 1, 3 and 4.
- An integrated system of sophisticated application programs: word processing, database management, communications, general ledger and more, all with powerful capabilities, surprising speed and ease of use.
- With source code, for custom modifications by you or MMS.
- The famous MMS support, including detailed manuals and examples, telephone tips, additional programs and inexpensive program updates, User Groups worldwide, the MMSFORTH Newsletter, Forth-related books, workshops and professional consulting.

mmsFORTH

A World of Difference!

- Personal licensing for TRS-80: \$129.95 for MMSFORTH, or "3+4TH" User System with FORTHWRITE, DATAHANDLER and FORTHCOM for \$399.95.
- Personal licensing for IBM PC: \$249.95 for MMSFORTH, or enhanced "3+4TH" User System with FORTHWRITE, DATAHANDLER-PLUS and FORTHCOM for \$549.95.
- Corporate Site License Extensions from \$1,000.

If you recognize the difference and want to profit from it, ask us or your dealer about the world of MMSFORTH.

MILLER MICROCOMPUTER SERVICES
61 Lake Shore Road, Natick, MA 01760
(617) 653-6136

CIRCLE 61 ON READER SERVICE CARD

SNOBOL, LISP, and OASIS

Dear Editor:

This is my second time on your BBS (first time was some months ago). I am very impressed with the first two issues of your magazine—keep up the good work.

I am also impressed with the concept of and the services available on your BBS. I would enjoy seeing some articles in *COMPUTER LANGUAGE* on SNOBOL replacements (ICON and SL-5) and on the new implementation of LISP, Common LISP. Also, I would very much like to see some coverage of the OASIS operating system from Phase One Systems and OASIS Technology. I use OASIS regularly, and I feel it is one of the best (albeit poorly known) operating system available for Z-80 micros.

Larry A. Schrupp
Colstrip, Mont.

He'll pay the bill

Dear Editor:

I ordered your magazine with the intention of taking the first issue and cancelling. Your ad plan worked; I am hooked. *COMPUTER LANGUAGE* has a "feel" unlike any other computer maga-

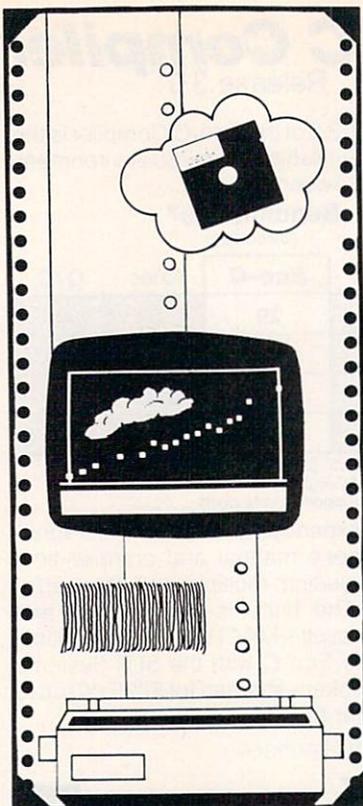


Illustration: Anne Doering

zine I've read. I'll still take *BYTE*, but I'm going to pay your bill when it comes.

Thanks for a good magazine and the Bulletin Board Service. Both are valuable to me. Your setting up this board is part of the philosophy that gives you that different feel.

Lance Reichert
Dayton, Ohio

Suggestions for the future

Dear Editor:

Thanks for an excellent magazine and for your progressive use of bulletin boards. I have a couple of suggestions for future issues:

- An EDITORS section. This could compare editors (vi, emacs, teco . . .) in the same manner in which you compared C and Forth in the first issue.
- A TOOLS section to look at things like Smart-C for automating programming tasks.
- A CODE LIBRARIES section that discusses things like the Greenleaf C library (not just public domain stuff).
- There are a lot of us Macintosh programmers out here looking for development support tools and information . . . domain stuff.

Ric Ford

THE WORLD'S FASTEST S-100 Z-80 SLAVE PROCESSOR

TurboSlave I

- 8 Mhz Z-80H
- Data transfers to 1 mbyte/second
- S-100 IEEE-696 compatible
- 4k Monitor rom
- Low parts count
- No paddle boards
- GUARANTEED COMPATIBLE WITH ALL S-100 SYSTEMS RUNNING TURBODOS
- 128k Ram with parity
- 2 RS-232 Ports. 50-38.k baud
- F.I.F.O. communications
- On board diagnostics
- Low power consumption
- TurboDOS compatible

INTRODUCTORY PRICE \$495

Includes TurboDOS drivers (a \$100 value) and TurboSlave I with 128k ram.



EARTH COMPUTERS

P.O. Box 8067, Fountain Valley, CA 92728
TELEX: 910 997 6120 EARTH FV

**FOR MORE INFORMATION AND QUANTITY DISCOUNTS
CALL: (714) 964-5784**

Registered trademarks: Z-80H, Zilog Inc.; TurboDOS Software 2000, Inc.
*** IBM PC VERSION COMING SOON ***

CIRCLE 24 ON READER SERVICE CARD

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



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 5 1/4 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.

CIRCLE 27 ON READER SERVICE CARD

We Do
Windows!

FORGET

OPTIONAL
8087
SUPPORT

EVERYTHING YOU THOUGHT YOU KNEW ABOUT PROGRAMMING IN BASIC.

introducing:

Better BASIC™

"It combines the best points of interpreted Basic, Pascal, Forth and Assembler... It's the first piece of software I'd spend my own money on."

Susan Glinert-Cole
Technical Editor
PC Tech Journal

BetterBASIC offers:

- Support of large memory (to 640K).
- Extensibility (Make your own BASIC!!)
- Speed. Sieve of Eratosthenes Benchmark:
 - BetterBASIC: 31.9 seconds.
 - IBM PC BASIC: 191.1 seconds.
- Program Block Structures.
- User defined Procedures and Functions.
- Local and Global Variables.
- Shared Variables.
- Recursion.
- Argument type validation.
- Optional arguments.
- Arguments passed by-value or by-address.
- Separately compiled program Modules.
- Simple interface to Assembly Language Procedures.
- Support for OEM hardware through extensibility.
- Useful set of Data Types:

- Byte, Integer
- Real (variable precision BCD) Ideal for business math.
- String (up to 32768 characters)
- Record Variables & Structures.
- N-dimensional Arrays of any type.
- Arrays of Arrays.
- Pointer (of any type)

General Information:

- Interactive programming language based on an incremental compiler.
- Syntax checked immediately on entry, with concise error reporting.
- Built-in Screen Editor allows on-line editing.
- Full IBM Graphics/Communications Support.
- Built-in Linker for separately compiled program Modules.
- Built-in Cross Reference Lister
- Built-in WINDOWS support!!
- 8087 math support

Computer Requirements:

- IBM PC, IBM PC/XT or compatible.
- PC/DOS 1.1, 2.0, 2.1
- 192K to 640K memory
- Usable on plain MS-DOS machines with reduced functionality.
(no Editor, Graphics or Windows)

OEM & Dealer Inquiries Invited.

BetterBASIC is a trademark of Summit Software Technology, Inc.
IBM PC, IBM PC/XT and PC/DOS are trademarks of International Business Machines Corp.
MS-DOS is a trademark of Microsoft Corp.

We are so sure you will like Better-BASIC, we will give you a 30-day money-back guarantee. Order BetterBASIC now!

BetterBASIC: \$199.00
8087 Module: \$99.00

Not convinced? Then try the Better-BASIC Sample and you will find that BetterBASIC is truly a major breakthrough in computer programming.
Sample disk: \$10.00



CALL YOUR DEALER OR SUMMIT SOFTWARE AT 617-235-0729

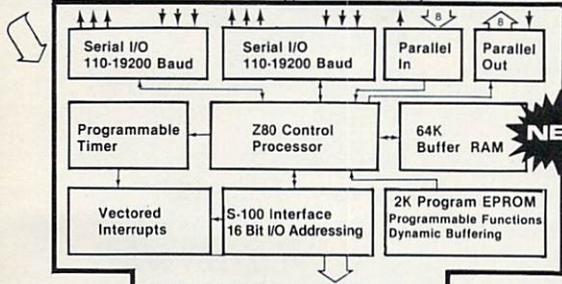
Summit Software Technology Inc.™ P.O. Box 99 Babson Park Wellesley, MA 02157

MasterCharge, Visa, P.O., Checks, Money Orders and C.O.D. accepted

CIRCLE 62 ON READER SERVICE CARD

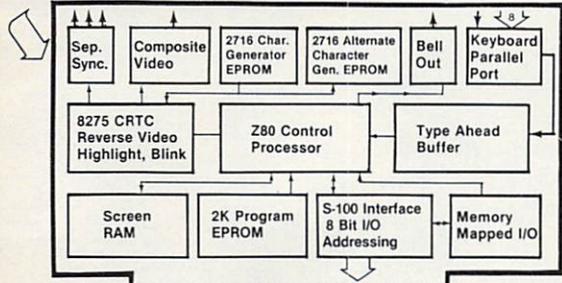
BUFFERED I/O BOARD

Introductory Price *\$59.95
With despool functions, protocols supported: XON/XOFF, ETX: ETB/ACK



80 CHARACTER VIDEO BOARD

25 Lines with status, compatible with Wordstar & dBase *\$49.50



Includes Bareboard, Heatsink & Documentation Call or write for more information.

Simpliway Products Co.
P.O. Box 601
Hoffman Estates, IL 60195
(312) 359-7337



OEM dealer pricing available, \$3.00 S/H, IL. Res. add 7% tax
dBase™ - of Ashton - Tate Corp. - Wordstar™ - of Micropro Int'l. Corp

CIRCLE 70 ON READER SERVICE CARD

YOU DON'T NEED EUREKA!™ ?? CONGRATULATIONS!!

We admire your talents. After all, few people can remember where to find that six month old letter to Wonder Waffle Works, or which of the twenty versions of IMPORTNT.BAS is the one you need yesterday.

Or maybe we should envy your spare time. Ah, to be able to haul out a stack of disks, slip each one into a drive, browse through the directory, and TYPE the various prospects to find that one file or program.

Or perhaps you're the adventurous type who thrills to the challenge of groping through scantily labeled disks, cheering that magical moment when hidden treasures are uncovered.

On the other hand, it occurs to us that you just may not know the advantages of EUREKA!, the fast, menu driven disk cataloger for CP/M. EUREKA! puts your entire disk library at your fingertips. Files may be found quickly and easily - by name or by comments you can put in the file itself. Of course the manual includes a tutorial to help you get started.

Still only \$50. Ask your dealer, or contact:

Mendocino Software Company, Inc.

Dept. L-1
P.O. Box 1564
Willits, CA 95490
Phone: (707) 459-9130

I found it!



Add \$2.50 Shipping:
Calif. residents add \$3.00 sales tax.

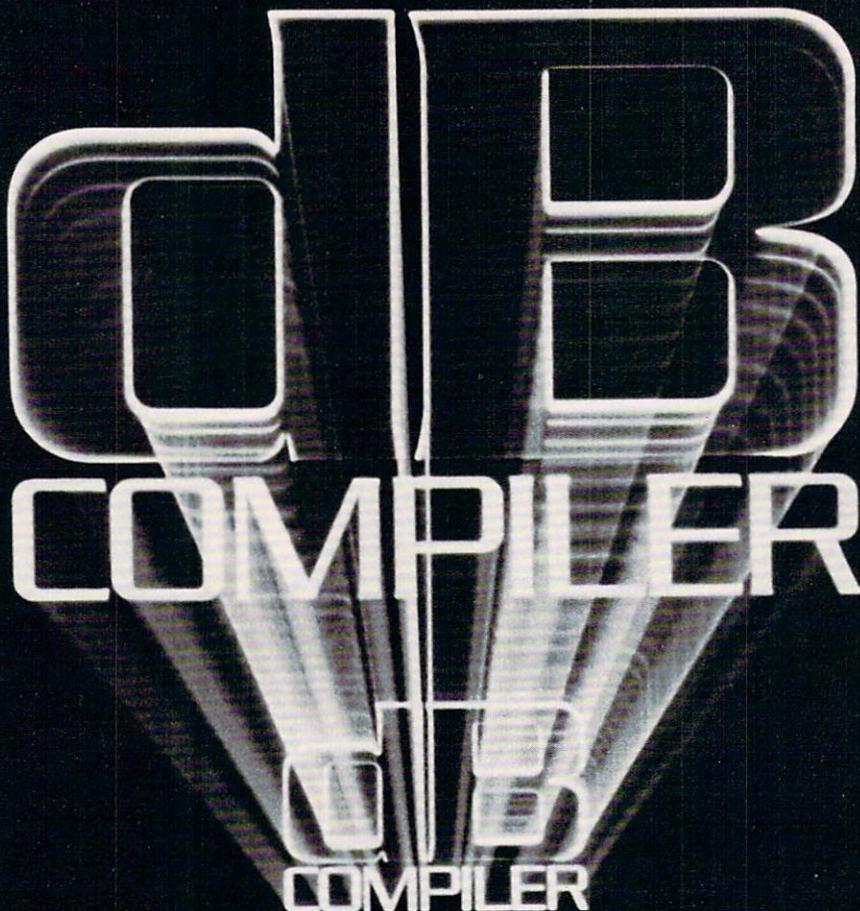
We accept VISA
& Mastercharge

CP/M is a registered trademark of Digital Research Corp.
EUREKA! is a trademark of Mendocino Software Co., Inc.

CIRCLE 48 ON READER SERVICE CARD

dB COMPILER

TM



The first compiler for dBASE II®

— SPEED —

dB Compiler™ produces applications which execute substantially faster than under dBASE II® in 16-bit environments. Some operations are even faster than under dBASE III®!

— INDEPENDENCE —

Buy dB Compiler™ once and compile and distribute as many applications as necessary with no additional cost. WordTech imposes no licensing fees, and a compiled application will execute without dBASE II or RunTime®.

— SECURITY —

Compilation is far better than encryption for protecting programming insights and procedures.

— PORTABILITY —

dB Compiler's™ cross-environment linkers make it easy to generate executable code for several operating systems.

For CP/M-80®, CP/M-86®, PC-DOS®, and MS-DOS®.

Suggested retail price: \$750; Cross-environment linkers: \$350.

Corporate/Multi-user licenses available.

dB COMPILER™

WORDTECH SYSTEMS, INC. P.O. Box 1747 Orinda, CA 94563 [415] 254-0900

CP/M-80, CP/M-86®, DRI PC-DOS®, IBM MS-DOS®, Micro-Soft Corp., dBASE II, RunTime, dBASE III®, Ashton-Tate, Inc.

CIRCLE 41 ON READER SERVICE CARD

BACK TO THE DRAWING BOARD

Great news! Within just a short time, *COMPUTER LANGUAGE* has become one of the most active SIGs on CompuServe. (You can access *COMPUTER LANGUAGE*'s SIG by typing "GO CLM".)

Now we have an "electronic soapbox" via CompuServe. You can hold forth (pardon the pun) on any subject you want and get immediate reaction. Although Back to the Drawing Board is concerned with problems and solutions, sometimes the black and white distinctions between what's called a problem and what's called an opinion gets a little gray. Let's look at some examples . . .

Jim Kyle, Tim Parker, Ron Bernstein, Barry Boland, John McNamee, Greg Law and I were part of the following conversation on CompuServe under the subject heading Language Wars.

Jim: C should be duck soup for any Pascal programmer! The structure is nearly identical. The major difference, as I keep telling my Pascal buddies, is the lack of strong typing—which is the feature that has kept me, as a systems-level programmer, away from Pascal.

Ron: With so many programmers writing COBOL code eight hours a day, one ought to think twice before PICing on its reputation.

Tim: If I get a few COBOL programmers insulted . . . great! We may get a little heat in the SIG after all. I still would like anyone to defend COBOL as a useful language!

Barry: Give COBOL credit where credit is due. There has to be something for everything else to be better than!

Greg: COBOL, the only program to require 300 lines of initialization per line of code. I programmed in COBOL once and found it a big mistake. I keep the card deck to remind me of that mistake! 800 cards to test memory. Redid it in assembly in about 75 cards. Talk about hassles.

Next is a thread of messages about a problem every programmer faces: how best to handle documenting right there in the source

code. Parties involved here are Pete Holsberg and Jeff Young.

Pete: It depends on the language. A self-documenting language like Pascal needs (my opinion) only an intro for the whole schmeer, for each procedure, and for each function. You know, "Converts the string in NAME to a . . ." kind of thing.

In assembly language, I like intro comments and lots of in-line comments. The trouble with assembly language is that most mnemonics aren't. One must work with a mnemonic set day in and day out to remember them well enough to figure out what the program is doing. I like the documentation that Intel has on the SDK-85 monitor program.

Now, if you always write in only one or two languages, and you do that constantly, then you will be able to read your own code in your sleep. I'm talking about non-programmer programming: the engineer who wants to write her own application program because there isn't any commercial program available. That person needs great documentation so she can debug the d---d thing!

Jeff: As a professional programmer/analyst, I frequently run across the question, "How much in-line documentation is enough?" The general approach that I and most of the programmers I work with have taken is to insert comments in front of each major section identifying the general function and then further define comments within the section as needed. For a non-programming type person, I would tend to agree with the concept that the program should be documented down to the detail code line. This type of person has a very limited knowledge of programming and could easily get confused as to the actual function of the code.

There is no real way to ensure that the comments have been updated, but if a program change alters the function of a section of code, most programmers will document it with some comment. Any changes that don't alter the function performed should not have to be documented in comments as the code itself would be sufficient.

Pete: But that's exactly the point! It just ain't good enough that "most programmers will document . . . with some comment." It is a fact that maintaining docu-

Electronic soapboxes

By Burton Bhavisyat

mentation is not enforceable; a program that works will work with (a) no comments, (b) good comments, (c) poor comments, and (d) wrong comments!

By the way, Dennis Sarris of A-FutreDat tells me that he runs a service of writing documentation (in either English or Spanish!) for anyone's programs. He takes your source code, studies it, and comes up with manuals. If you think that concept can release you from some struggling, contact me here at *COMPUTER LANGUAGE*, and I'll line you up with his address.

Getting back to the documentation in source code issue, here's a conversation held recently between programmers John McNamee, Jim Kyle, David Zook, Tim Smith, Pete Holsberg, and Cheyenne Wills.

John: Have you ever seen the source code to the UNIX shell? The guy does lots of things like . . .

```
#define FOREVER for (;;)
#define IF if (
#define THEN ) {
#define ELSE } else {
#define FI }
```

You get the idea. That program doesn't look at all like C, and it is impossible to read. I'm not sure advocating defining FOREVER is such a good idea. The all caps looks terrible in a sea of lower case, and any good C programmer should be able to see *for (;;)* and know what it means (if they can't handle that, they shouldn't be using C).

Jim: Gotta disagree with that last statement, John. It's like saying that anybody who can't drift through a 90-degree bend at 95 mph shouldn't try to learn to drive! The all upper-case #defines are recommended by Kernighan (in *Software Tools* and also in *K&R*) specifically to indicate that what you see is not what the compiler will get, and FOREVER indicates what you were trying to do, while *for(;;)* looks

like you left something out, unless you already know why it's being done.

I have learned the hard way to comment my code extensively and make maximum use of mnemonic variables and defines. When I have to come back next year and maintain it, I'm not the same person I was when I wrote it and need all the help I can get in remembering why I wrote it the way I did. As Kernighan (or maybe Plauger) wrote in *The Elements of Programming Style*, "If you're as clever as you can be when you write the code, how are you

ever going to be able to debug it?"

John: My point is that a program full of upper-case defines can end up looking ugly. The worst case of this is the UNIX shell. Only S.R. Bourne, the author, likes the way it looks. It's a mess for the rest of us who have to maintain it.

Jim: In which case, why not take a good editor to your own copy and sweep through, replacing the *IF*, *THEN*, *ELSE*, etc., references with what they stand for? I agree with you so far as overusing the *define* capability to make C look like some

other language goes, but if it's not made meaningless by overuse, the special-definition technique can add a whole new dimension of readability to a program.

David: I agree with using *FOREVER* and other things to make your intent clear. The more I have to mentally combine several lines of code into one comment that tells what the code does, the worse the language (or the programmer) is. An ideal language can be read top-down, without bottom-up reconstruction of the programmer's intent (as is necessary with *while(1)* or *for(;;)*). C is a long way from being an ideal language, but *define* lets you make it more palatable.

Pete: I agree with you 100%!

Jim: The UNIX shell was written by Steve Bourne, who is (was) quite an ALGOL fan. As a C programmer, I find it very hard to read.

Tim: *for(;;)* is such a common C idiom that there is no reason to rename it.

Pete: But that's just the point: it's idiomatic, therefore less than obvious.

John: But *for(;;)* is so common that any good C programmer should know it. Use *#define* on the hard stuff and leave the basic idioms to the mind of the programmer (any C programmer who can't see *for(;;)* and know what it means probably doesn't have a mind). You have to have seen a program full of new syntax for basic things before you can appreciate how much it gets in the way of understanding the program.

Cheyenne: Just to put my two cents in. When using *define*s and such you are actually redefining the language. That may be fine for you, but think of the person who must then maintain that program. He may know C inside and out, but now he has to learn a new "language." Having been a systems type for some time, working mostly with assembly language, I've come across "assembly" programs that could have been compiled by a PL/I compiler. The number of macros that were used made the program much harder to maintain. Plus, think of it the other way around.

If you have been redefining the language and you have someone learning from that code, that person will be hindered later when he must maintain other code. Not that I'm against using macros—they have their place.

John: That's what I've been trying to say, but very few people seem to agree.

Tim: Pete—All human languages are irregular and complex to some degree. But let's define terms here. An idiom is a phrase whose meaning cannot be directly determined from its constituent words.

In English, the phrase "kick the

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, INC.

2101 HICKORY DRIVE ◆ CARROLLTON, TX 75006 ◆ (214) 446-8641

CIRCLE 44 ON READER SERVICE CARD

bucket" meaning to die is idiomatic. So *for(;;)* in C is idiomatic only in a weak sense. You can figure out what it means from the language definition, but its meaning is not immediately obvious. I don't think a computer language with a lot of idioms would be very useful, but most computer languages have constructions that are idiomatic in this weak sense, and we just have to learn them to be effective users of the language. Alas, nothing is perfect.

If you've been wondering how to get at all the wonderful data bases full of languages and utilities as mentioned all over the SIG, here's your answer in an exchange between Dan Martin and Tim Parker and a little goodie from Eli Willner.

Dan: So far I've spent about 30 min (\$\$\$) trying to actually find the data bases that are so freely referenced. What menu path do I take to get into them and look at their contents?

Tim: This version of the SIG program doesn't have the menu selection for the data bases, which has led to a bit (!) of confusion. To enter the data bases, type XA # where # is 0 to 10. Then menus appear. Subject headings can be seen by typing "SN" at the main menu.

Eli: This information may help you in your quest for public domain languages.

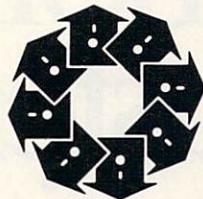
A good source of information is the Digital Equipment Computer User Society (DECUS). This user group is partially sponsored by DEC and has a large software library, all of which is public domain. The library has compilers for Pascal, APL, ALGOL, RATFOR, LISP, C all available in source form. Some are specific to DEC machines, of course, and are written in DEC assembler language. But many are written in high-level languages and can be adapted to other environments.

The UCSD Pascal Users Society (USUS) maintains a library of software which is not public domain but is available to members for non commercial use for a nominal fee (usually on the order of \$5 to \$10 per diskette). This library contains an Ada syntax checker, a LISP interpreter, and an APL interpreter, among many other useful programs. These are written in Pascal.

Membership in DECUS is free; membership in USUS costs \$25 per year. Those interested in joining USUS can check out MUSUS, our SIG here on CompuServe (R SIGUSC from this SIG).

An early version of the UCSD Pascal compiler (version 1.3), which is fully

**UNPARALLELED
PERFORMANCE
and
PORTABILITY
in an
ISAM PACKAGE
at an
UNBEATABLE
PRICE**



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:
8" CP/M® 5¼" PC-DOS 8" RT-II

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

Access Manager and CP/M are trademarks of Digital Research, Inc.
c-tree and the circular disc logo are trademarks of FairCom

©1984 Faircom

CIRCLE 29 ON READER SERVICE CARD

**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 **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 program 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 \$175.00 from DWB Associates or ask your software dealer. To order or for more information, call or write DWB Associates. VISA/MC accepted. Dealers welcome.

IBM is a trademark of IBM Corp. MSDOS is a trademark of Microsoft Corp.
THE PROFILER is a trademark of DWB Associates.



P.O. Box 5777
Beaverton, Oregon 97006
(503) 629-9645

CIRCLE 20 ON READER SERVICE CARD

functional but lacks the enhancements of the current version (separate compilation, concurrency, etc.), was developed under a government grant and is also public domain. This compiler generates p-code but can be adapted without much difficulty to generate native code instead (in fact, many current commercial Pascal compilers used this implementation as their base).

This compiler (which is in Pascal) is available on the PDP-11 SIG on CompuServe (R SIGM11 from this SIG). Membership in the PDP-11 SIG is free;

just leave a message to Sysop with your full name, asking for membership. You will then have access to the on-line software library.

I'm most familiar with the DEC and p-system worlds. I'm sure there are plenty of other PD languages floating around for other machines and OSs as well.

 Looking through my mailbox this month, I came across an item of great interest to developers of software for the MAC. The Pro-

grammer's Shop, Hanover, Mass., has installed a bulletin board system (similar to the one here at *COMPUTER LANGUAGE*) at (617) 826-4086 to be used as a forum for discussion related to MAC development.

Among topics being discussed are development software (for the MAC) running on an IBM XT (really, it says that right here), a newsletter, a BASIC compiler, several C compilers, a Modula-2 compiler and miscellaneous utilities. There is no charge for participation in this exchange.

If you've been following the action here on Back to the Drawing Board and on the *COMPUTER LANGUAGE* BBS, you know that we've assembled a group of experts who've volunteered to answer your questions. Every so often, I'll do a little profile on one or two of our volunteer experts.

Dr. Timothy C. Prince, Marblehead, Mass, says, "I would like to volunteer for your list of 'experts.'" I have a lot of experience with FORTRAN, medieval and modern, and RATFOR preprocessors on a variety of machines from IBM 7090 to Z80 to 68000 to Floating Point Systems 164.

"I have done quite a bit of patching Z80 code over 8080 code in WordStar and the CP/M BDOS and have developed BIOS code for CP/M 2.2 as well as patches to make non-2.2 .COM files run.

"I've been doing work on FORTRAN to C conversion in order to take advantage of matrix analysis software. Good Luck with your project!"

And from Art Winston, Fort Lauderdale, Fla., comes, "my high touch contribution is knowledge about commercial stock market and commodity data bases such as Quicktrieve and Dow Jones and how to get at them using BASIC, FORTH, and SAS."

Thanks go to Winston and Prince and all our volunteer experts for helping to make the *COMPUTER LANGUAGE* expert forum a success. 

For your IBM/PC

mbp COBOL: 4 times faster, and now with SORT & CHAIN.

\$750.

mbp COBOL can be summed up in one word: fast.

Because it generates native machine language object code, the mbp COBOL Compiler executes IBM/PC* programs at least 4 times faster (see chart).

allow source & object code, map & cross-reference checking; GSA Certification to ANSI '74

Level II; mbp has it all.

It's no surprise companies like Bechtel, Chase, Citicorp, Connecticut Mutual, and Sikorsky choose mbp COBOL; make it your choice, too. mbp is available at Vanpak Software Centers, or direct. For complete information, write **mbp Software & Systems Technology, Inc.**, 7700 Edgewater Drive, Suite 360, Oakland, CA 94621, or phone 415/632-1555 -today.



GIBSON MIX Benchmark Results

Calculated S-Profile
(Representative COBOL statement mix)

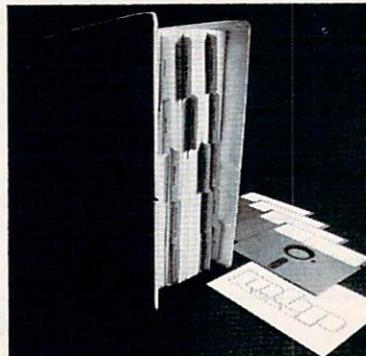
Execution time ratio

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 is an IBM TM; **Level II is a Micro Focus TM; ***A Ryan-McFarland TM; ****A Microsoft TM.

Fast also describes our **new SORT**, which can sort four-thousand 128-byte records in less than 30 seconds. A callable subroutine or stand-alone, 9 SORT control fields can be specified. And our **new CHAIN** is both fast and secure, conveniently transferring control from one program to another, passing 255 parameters. Plus, **new extensions** to ACCEPT & DISPLAY verbs give better, faster interactive programming.

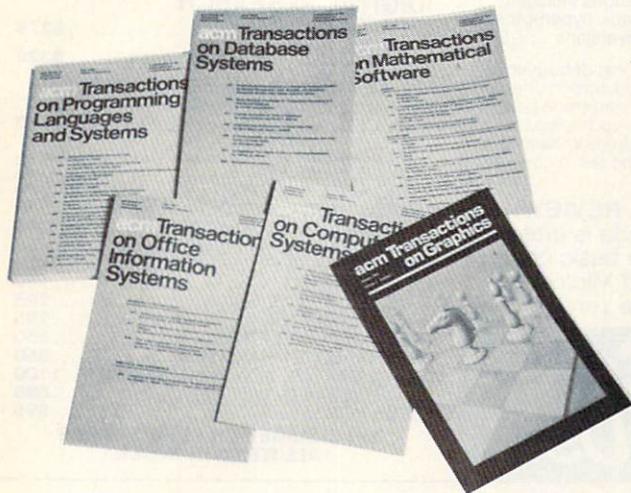
The complete COBOL. An Interactive Symbolic Debug Package included standard; Multi-Keyed ISAM Structure; listing options



CIRCLE 39 ON READER SERVICE CARD

Journals you read cover to cover

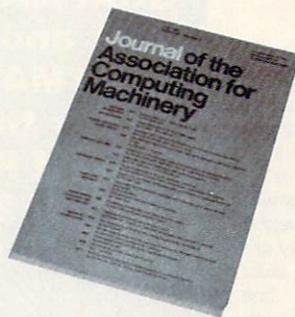
From computing applications to computing theory, from matters of practical importance to those of scientific research, ACM journals offer you high quality informative articles. Each undergoes a thorough review to insure its accuracy, topicality, and pertinence. Every journal is specially tailored to serve the specific needs of the computing community.



source for comprehensive surveys, tutorials, and overview articles on topics of current and emerging importance. The *Journal of the Association for Computing Machinery* presents fundamental ideas that are of lasting value to the understanding of computation.

ACM also publishes unique reference sources. *Computing Reviews* contains original reviews and abstracts of current books and journals. The *ACM Guide to Computing Literature* is an important bibliographic guide to computing literature (available annually on Standing Order Subscription). *Collected Algorithms from ACM* is a collection of ACM algorithms available in printed version, on microfiche, or on machine-readable tape.

While *Communications of the ACM* has made its mark publishing landmark research papers in computer science, today the magazine is moving into a broader overview role. The editorial aim is to publish broad-gauge, high quality, highly readable articles on key issues and major technical developments in the field. The various transactions (*ACM Transactions on Mathematical Software*, *ACM Transactions on Database Systems*, *ACM Transactions on Graphics*, *ACM Transactions on Programming Languages and Systems*, *ACM Transactions on Office Information Systems*, and *ACM Transactions on Computer Systems*) cover burgeoning areas of computer research and applications. *Computing Surveys* is your



CIRCLE 7 ON READER SERVICE CARD



For further information about ACM journals, send for your free ACM Publications Catalog today!

- Please send me a free ACM Publications Catalog.
- Please send me subscription information for the following journal(s):

<input type="checkbox"/> Communications of the ACM	<input type="checkbox"/> ACM Transactions on:
<input type="checkbox"/> Journal of the ACM	<input type="checkbox"/> Mathematical Software
<input type="checkbox"/> Computing Reviews	<input type="checkbox"/> Database Systems
<input type="checkbox"/> Computing Surveys	<input type="checkbox"/> Programming Languages & Systems
<input type="checkbox"/> Collected Algorithms from ACM	<input type="checkbox"/> Graphics
<input type="checkbox"/> ACM Guide to Computing Literature	<input type="checkbox"/> Office Information Systems
	<input type="checkbox"/> Computer Systems

- Member No. _____ Expiration Date _____
- I am not an ACM member, please send me membership information.

Name _____

Address _____

City _____ State _____ Zip _____

Mail to: Publications Department,
ACM, 11 West 42nd Street, New York, NY 10036.

THE 8087 AND 80287 ARE IN STOCK!

REAL TIME MULTI-TASKING/ MULTI-USER EXECUTIVE - RTOS

RTOS is a MicroWay configured version of iRMX-86. Includes ASM-86, LINK-86, LOC-86, LIB-86, and the ROM Hex Loader. \$600

87FORTRAN/RTOS™ - our adaptation of the Intel Fortran-86 Compiler generates in line 8087 code using all 8087 data types including 80-bit reals and 64-bit integers. The compiler uses the Intel large memory model, allowing code/data structures of a full megabyte, and supports overlays. Includes RTOS and support for one year \$1350

RTOS DEVELOPMENT PACKAGE includes 87FORTRAN, 87PASCAL, PL/M-86, Utilities, TX Screen Editor and RTOS. \$2500

OBJ→ASM™ - a multipass object module translator and disassembler. Produces assembly language listings which include public symbols, external symbols, and labels commented with cross references. Ideal for understanding and patching object modules and libraries for which source is not available \$200

MATRIXPAK™ manages a MEGABYTE! Written in assembly language, our runtime package accurately manipulates large matrices at very fast speeds. Includes matrix inversion and the solution of simultaneous linear equations. Callable from MS Fortran 3.2, 87MACRO, 87BASIC, and RTOS each \$150

CALL FOR COMPLETE CATALOG

87BASIC/INLINE™ generates inline 8087 code! Converts the IBM Basic Compiler output into an assembly language source listing which allows the user to make additional refinements to his program. Real expression evaluations run seven times faster than in 87BASIC. \$200

87BASIC™ includes patches to the IBM Basic Compiler and both runtime libraries for USER TRANSPARENT and COMPLETE 8087 support. Provides super fast performance for all numeric operations including trigonometrics, transcendental, addition, subtraction, multiplication, and division \$150

87MACRO™ - our complete 8087 software development package. It contains a "Pre-processor," source code for a set of 8087 macros, and a library of numeric functions including transcendental, trigonometrics, hyperbolics, encoding, decoding and conversions \$150

87DEBUG™ - a professional debugger with 8087 support, a sophisticated screen-oriented macro command processor, and trace features which include the ability to skip tracing through branches to calls and software and hardware interrupts. Breakpoints can be set in code or on guarded addresses in RAM \$150

PC TECH JOURNAL REVIEW:

"The MicroWay package is preferable... it executes the basic operations more rapidly and MicroWay provides a free update service."

8087-3 CHIP..... \$149
including DIAGNOSTICS and 180-day warranty
64K RAM Set..... \$35
256K RAM Set..... \$250
80287 CHIP..... \$350
8087 8mhz..... \$375

PC AT 30 MEGABYTE WINCHESTER DRIVE..... \$2000

MICROSOFT FORTRAN 3.2..... \$239

MICROSOFT PASCAL 3.2..... \$209

These IEEE compatible compilers support double precision and the 8087.

DIGITAL RESEARCH FORTRAN..... \$279

LATTICE C with 8087 support..... \$329

FFT87 an FFT package for the 8087. Does Forward and Inverse Transforms on complex data. Callable from SSS or MS Fortran ... \$150

IBM Basic Compiler CALL

Alpha Software ESP 595

STSC APL★PLUS/PC 500

TURBO PASCAL or SIDEKICK 45

TOOLBOX 45

TURBO PASCAL with 8087 Support 85

HALO GRAPHICS CALL

GRAPHMATIC 125

ENERGRAPHICS 295

Professional BASIC 295

COSMOS REVELATION 850

MAYNARD WS1 HARD DISK 950

MAYNARD WS2 or WS3 HARD DISK 1109

smARTWORK by WINTEK 895

SPSS/PC 695

NO CHARGE FOR CREDIT CARDS

ALL ITEMS IN STOCK

MicroWay

P.O. Box 79
Kingston, Mass.
02364 USA
(617) 746-7341

**You Can
Talk To Us!**

CIRCLE 57 ON READER SERVICE CARD

Total Support Packages

The GRAFMATIC (screen graphics) and companion PLOTMATIC (pen plotter) libraries of modular scientific/engineering graphics routines let you easily create 2D and 3D plots in customized or default formats. Pen plot preview with GRAFMATIC. Plot interactively or in deferred mode. Others only provide our "primitives" (mode, color, cursor, character, pixel, line, paint...). We follow through with: auto-scaling, auto-axis generation, auto-tic mark labeling, function plots, tabular plots, auto-function plots (complete plot in default format with one easy call), auto-tabular plots, log/parametric/contour plots, 3D rotation/scaling/translation, wire frame model (for old time's sake), hidden line removal for solid models (GRAFMATIC only), cubic and bicubic spline interpolants, least squares fits, bar and pie charts, screen dump.... You name it, We have it! Best of all, the clearest and most complete documentation to be found in microcomputerland. User support? Of course, call us! We offer a no questions asked money-back guarantee.

**GRAFMATIC™
and
PLOTMATIC™
for the
IBM PC
Tandy 2000
TI Professional**

**FORTRAN
PASCAL
Screen and
Pen Plotter
Graphics
Tools**

MICROCOMPATIBLES

11443 Oak Leaf Dr
Silver Spring, MD 20901
(301) 593-0683

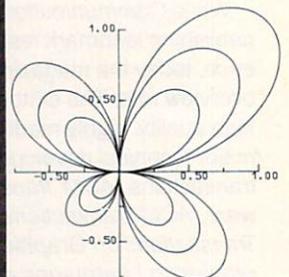
GRAFMATIC \$135

PLOTMATIC 135

BOTH 240

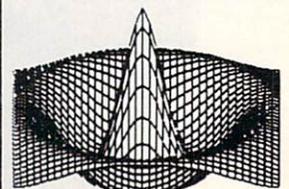
Specify compilers:
(IBM/MS/SuperSoft/Digital
Research)

Plotters: (H-P, HI, IBM)



"... GRAFMATIC is the most imaginative and well designed use of FORTRAN I have yet seen in a FORTRAN microcomputer software package."

James Creane,
Contributing Editor,
Personal Computer Age



CIRCLE 15 ON READER SERVICE CARD

The art of hacking

By Ken Takara

In the October 1984 issue of *COMPUTER LANGUAGE*, we presented the formal approach to software design. This time, let's look at the other side—the world of the software hacker.

To some, a hacker is merely a sloppy and disorganized programmer. To others, they are highly skilled craftsmen. The general public often associates the term with computer vandals.

This month, dedicated hackers Dale Carlson, Chris Wiley and Dave Pifer describe their techniques and philosophies, providing some insight into that much maligned practice of hacking.

Carlson is a professional engineer who works with real-time processes such as motor controls. Among the projects he has done is the firmware for a microprocessor-driven burglar alarm system.

Wiley prefers to program non-professionally after dark. He is the author of *PC Musician*, a user-supported program for the IBM PC. He considers hacking a high art not found in the production environment.

Pifer's programs are generally written for personal use first and then often published in magazines. He is the founder of Computer Hobbyists Against Raiders and Thieves (CHART), a section of the Writer's and Editor's Special Interest Group on CompuServe. This organization is dedicated to regaining the dignity that hackers have lost due to the media's misuse of the term.

First off, what is hacking? Or, perhaps, what is the difference between a designer and a hacker?

Carlson: A designer makes plans while a hacker makes programs. A designer will lay out the entire project before doing anything else, trying to anticipate everything before it ever comes up. A designer will produce reams of paper, specifications, designs, and so on, describing the project in intimate detail without ever producing anything solid.

A hacker also will try to get a handle on the problem as a whole, but will build a

core upon which the rest of the program can be built. Rather than try to anticipate it all, a hacker will tackle the problems as they arise.

I like to think of hacking as interactive programming—you build the program, interacting with it a lot.

Wiley: For me the important feature is that hacking is an individual thing. It's not a suitable approach for a group project.

But I think it's the most efficient way for a single person to program. This individuality makes hackers difficult to manage. They don't usually fit into a group project too well.

Hacking is highly creative programming, something like writing a novel. I don't think you can truly hack an assigned project because you lose the essence of the creative part: dreaming it up in the first place.

This business of calling computer vandals or burglars hackers is truly appalling. There is a romance to hacking, just as there is to writing novels. But I prefer to emphasize the positive, creative aspects.

Pifer: Hackers are designers. It's just the actual design technique that differs. Typically, a hacker will have spent countless hours going over a project before the first line of code is hacked. However, the hacker's planning usually takes place away from the computer. And almost never will the hacker's planning involve note taking.

True designers sit down for design sessions. They set aside time to think about the design and nothing else. Copious notes will be made and, more importantly, actually referred to while writing the program. I doubt seriously that a designer spends any more time in designing a program than does a hacker. It's just a more conscious effort.

As for preserving the reputation of the true hacker, I can't say enough against the misuse of the term. CHART has been very active in correcting the media when it uses hacker to describe computer criminals.

How do you approach a project?

Wiley: When I program something, I first of formulate at least a set of core

requirements of what I want the program to do. Then I see if I can get it to do that much. I generally use the first algorithm that comes to mind and see where it leads.

Once I get it to do the basic function I've visualized, I can stop and see what further functions or stylistic changes are suggested by using it. So the planning for me very rarely takes place on paper at all. It usually is a matter of thinking about what I want it to do and how I want it to interact in a general way. After that, it's a matter of picking the core functions and constructing a skeleton, then hanging the embellishments onto it.

I don't do any diagramming. Sometimes I'll use a little bit of pseudocode. Generally I can work just as fast by coding directly in C.

Carlson: Usually someone comes to me with a request for a program but doesn't really know what he wants. He'll give me some loose specs and say, "Give me something like this."

If I ask, "How do you want it done?" he'll say, "Do whatever you want to, as long as it works."

So then I have to figure out as closely as possible what he's after. I have to ask questions to get as much information as I can. Then I build a simple prototype that I can show him. It usually helps people to see something. Then he can say, "Oh, change this, add that." And so on.

Once I have an idea of what someone wants, I try to look at the program in its entirety. It's kind of a Gestalt approach to programming. You look at the big picture, then try to break it down into smaller blocks—usually the functional units. Finally I select a block and start coding.

Pifer: All of my projects fall into one of two categories. Either they are short-term, immediate-need programs or long-term, procrastination prone pet projects. Short-term projects involve peeling concepts from my long-term mental log and applying them to the problem at hand. Basically, if you keep enough long-term projects in mind you can build a powerful library of program modules.

The pet projects get a lot more planning put into them. They are nice because they give you something to do while driving to work, or while on hold, or during long meetings. Pet projects are almost always written in my mind before I start them. All that remains is to add bells and whistles and, of course, debugging.

Wiley: Hacking is characterized by its interactive nature. You try something, see if it works, and fix it till it does. I code the main module first, with a few of the core functions, testing them one at a time to see how they work together. Then I either optimize them to work together better or I add other functions.

The danger of coding the way that I do is that you can come up with unforeseen side effects. This is particularly true if you use a lot of global variables. Having been a BASIC programmer, I probably use more global variables in C than I ought to, leading to some bugs to squash.

Carlson: It's important to know how to modularize the program since you obviously can't take the whole thing on at one time.

The most important thing, though, is to select the right block to start with. You want to start with a block that is central to

the program. It will become the core upon which you will add everything else.

Pifer: The true beauty of hacking is that there is no such thing as wasted time or energy. Even if you've written half the program and have to scrap it for a new approach, you know that you will always use the scrap code somewhere down the road.

 **W**hat kind of language do you feel is conducive to hacking?

Carlson: The way I work, it helps to be able to test the core as you build on it. I build, then I like to test before adding more. I can hack in assembler, but it's not easy. I have to write extra code to test the routines.

A language like Forth is a hacker's paradise. You can build a small amount then test it without putting in a whole lot of effort.

Wiley: I used to hack in BASIC on an Apple II. There I ran into the phenomenon that I couldn't read my own code after a while.

Once I moved into the IBM PC world and got a C compiler, I suddenly found that I could hack and get away with it.

The reason is that the modularity of C

imposed a structure itself such that I could break my program down into functions and subroutines that were relatively small and easy to understand.

I've used Pascal. I'm inherently on the lazy side, I guess. I found all the declarations and rigamarole at the beginning of a procedure to be very tedious. With C, I can dive in with the function, then later go back and declare variables that I've used.

Forth is totally incomprehensible, and I've never owned a reverse polish calculator.

Pifer: Of course, BASIC is the ultimate hacking language. You have to respect a language that allows you GOTO GOSUBs and get away with it. But languages such as Pascal and C are also ideal. It goes back to my scrap theory. It is much easier to build a collection of blocks and then read them in as needed for any given project. Someday this collection of program blocks may make my mental notebook obsolete.

 **S**o in hacking you tend to use the build-as-you-go approach rather than the well laid out design-first approach?

ARTIFICIAL INTELLIGENCE

Programming: Learn Fast, Experiment, Prototype

with the nonprocedural language chosen by Japan

PROLOG-86™

implements the "standard", features described in Clocksin and Mellish, has tutorials and sample programs to learn from that include:

- an Expert System
- a Natural Language Interface

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. Become familiar in **one evening**, comfortable in days.

CONTEST: "Artificial Intelligence Concepts". All entries that teach key concepts and are clear will get recognition. The best will win \$1,000. Submit by 2/28/85.

AVAILABILITY: PROLOG-86 runs on MSDOS, PCDOS or CPM-86 machines. We provide most formats. The Intro price is \$125.

Full Refund if not
satisfied during
first 30 days.

617-659-1571

**Solution
Systems™**

335-L Washington Street
Norwell, MA 02061

CIRCLE 60 ON READER SERVICE CARD

RP/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. RMPPIP 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 RMPPIP.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

Carlson: Well, you need some sort of design. If you were to really hack without any plan at all, you wouldn't get anywhere. I don't think any hacker is that disorganized. Actually, it's a matter of degree.

I suppose there are some things you can do without any plan, while other things require a great deal of planning.

Wiley: In my experience, my approach of building a skeleton, then hanging the flesh and organs from it, hasn't resulted in my needing to rebuild the whole thing. It's a matter of letting the program evolve, rather than having the perspicacity to sit down and imagine in detail what it will end up as.

I started one program with a general idea of what it should do, then I planned the user interfaces. After that, I had to set up the data structures so that they could be manipulated the way I wanted. That was the extent of my design. Then it was a matter of sitting down at the keyboard and coding.

A lot of designing takes place at the subconscious level, in a dream-like state. I generally keep it in my head. I might draw a rough diagram of the way the screen looks, or I might make a list of

commands just to see that they make sense.

I generally design with my eyes closed, reclining in a chair just thinking about it.

Pifer: As I said before, I believe the hacker is always very well prepared when he sets out to write the program. Personally, I am most careful with the data structures within my programs. My reasoning for this is that as long as I know where and what the data is, there are 1,001 ways to go about getting to it.

Carlson: One problem with designers is the tendency to over-design.

Sometimes a designer will try to anticipate absolutely everything, and that's just impossible. Sometimes it turns out the world doesn't turn the way you thought it did when you laid out the design.

In real-time applications, many things don't show up until you're actually running the program. There are all sorts of hardware dependencies that you may or may not have anticipated.

Depending on how detailed your design is, you may have to go back and change the whole thing. You can end up with twice as much work. Or worse yet, you decide to skimp on the documentation and work on the code, and it all becomes outdated.

Pifer: An important difference between the designer and the hacker is

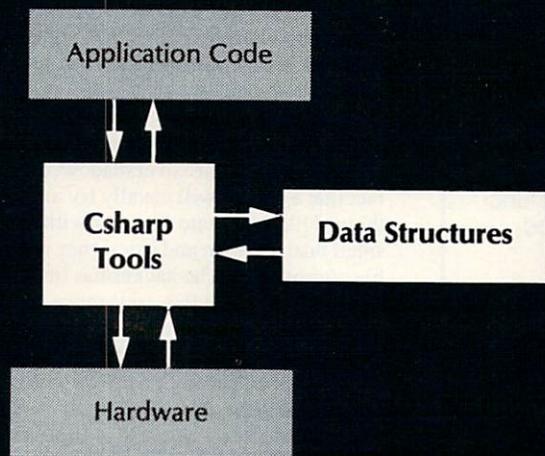
that once the designer attains the design, the program is complete. A hacker's program is never complete. Even when the program is finally debugged and running flawlessly, the hacker will always find a reason to go back in and change or enhance something.

I think this is the primary reason hackers tend to not document or to poorly document their programs. Only lately have I started to fully document my works. I always knew that I should but always postponed it until after the next version. Proper documentation is something that even the commercial developers have not caught on to yet, so how can we expect it of a hacker?

Wiley: I don't document my code. I find if names are long and descriptive enough, just looking at the code is sufficient to remind me of what it does. I find C is very friendly for that.

I think it would be impossible to figure out what was happening in Forth after putting it down for a day. I looked over some code once that was designed and well-commented. I did not find what it was doing immediately obvious. I still had

Csharp Realtime Toolkit



C source code for realtime data acquisition and control. Tools include: graphics, event handling, scheduling, and state systems. Processor, device, and operating system independent. \$600.00

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

CIRCLE 76 ON READER SERVICE CARD

C UTILITY LIBRARY

The **C UTILITY LIBRARY** is a set of **200+** functions designed specifically for the PC software developer. Use of the Library will speed up your development efforts and improve the quality of your work.

- BEST SCREEN HANDLING AVAILABLE
- WINDOW MANAGEMENT, COLOR GRAPHICS
- DOS 2 DIRECTORIES, COMMUNICATIONS
- KEYBOARD, PRINTER, TIME/DATE
- EXECUTE PROGRAMS, BATCH FILES
- STRINGS, BIOS, AND MUCH MORE
- ALL SOURCE INCLUDED—NO ROYALTIES

Available for Microsoft/Lattice \$149, Computer Innovations \$149, Mark Williams \$149, DeSmet \$99. Add \$3 shipping. N.J. residents add 6% sales tax. Visa, MC, checks—10 days to clear. Order direct or through your dealer. Dealer/Distributor inquiries welcome.

ESSENTIAL SOFTWARE, INC.
(914) 762-6605
P.O. Box 1003
Maplewood, N.J. 07040

CIRCLE 28 ON READER SERVICE CARD

to go over it a few times before I could understand it.

Carlson: I prefer to document what I have now, rather than what I think I'll have. There is a need for documentation, of course, but there's no point in making extra work for yourself.

A set of notes to remind yourself of what you're doing is useful. A whole book describing code you haven't even written yet is unnecessary.

Of course, I have to write some sort of manual for the user.

Wiley: If it's a general purpose routine that I'll be using in the future, then I do document it. But that's probably the only time. I don't write many general sub-routines, though. Most of my code is specific to the context of the application. If I need to write general utilities, it's because of a deficiency in the C library.

I have written manuals for the user. My personal feeling is that a program that leads you too much with many menus and prompts slows you down once you understand how to use it. Since I prefer a com-

mand driven program, user documentation is required.

Do you find any limitations to hacking?

Carlson: There is a limit to the size of the project you can take on by hacking. The bigger or more complex it is, the more planning you have to do.

You have to plan more carefully as more people are involved also. It's difficult for two or more people to hack a program together.

When you have a very complex program or a large team, you need to have someone to break the thing down into reasonable blocks or tasks. This person should also design the task interfaces so that everyone involved can talk to everyone else.

But you know, once the designer has done the task specs and designed the task interfaces, someone has got to build it. And that's probably a hacker.

Wiley: My own philosophy of software is of the school that favors small, tight, well-crafted tools rather than the big blunderbuss, do-everything programs. I doubt that I would ever tackle anything too big to think about or keep in my head.

I think this approach results in programs that are more clearly focused and ultimately more useful.

I think a lot of the truly great breakthrough programs are hacked. They may be eventually torn down and redone by committee for a finished version, but I'm sure that something like the first version of VisiCalc was hacked.

It's very hard to imagine any committee coming up with something new and original. That always springs from one creative individual.

Pifer: There are a number of limitations to what can be hacked. But all of these restrictions are overshadowed by the fact that a hacker will usually try anything. I like to equate a hacker with a small businessman and a designer with a big corporation. The hacker has little to lose if an idea falls flat, so is more likely to ignore the fact that what he wants to do can't be done. Fortunately for all of us, there have been enough hackers to keep this industry moving.

Next month we will take a look into the world of the mainframe as we bring to these pages a debate that has been brewing for nearly eight years: the next ANSI standard for FORTRAN, known as FORTRAN 8X. With a committee comprising approximately 40 primary members from a widely varied FORTRAN community, the number of diverging viewpoints can easily be a factorial function. ■

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*.

Find out how **Instant-C™** is changing the way that programming is done. **Instant-C™** is \$500. 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 (Intell Corp.), CP/M-86 (Digital Research, Inc.), Instant-C (Rational Systems, Inc.)

CIRCLE 56 ON READER SERVICE CARD

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 300 products

"C" LANGUAGE

	LIST PRICE	OUR PRICE
MSDOS: C86-8087, reliable	\$395	call
Instant C - Inter., fast, full	NA	500
Lattice 2.1 - improved	500	call
Microsoft C 2.x	500	349
Williams - NEW, debugger	500	call
CPM80' Ecosoft C - now solid, full	250	225
BDS C - solid value	150	125
MACINTOSH: Full, ASM	NA	385

Compare, evaluate, consider other Cs

BASIC

	ENVIRONMENT	PRICE
Active Trace-debug	86/80 NA	75
BASCOM-86 - MicroSoft	8086	395 279
BASIC Dev't System	PCDOS	79 72
BASICA Compiler - BetterBASIC - 640K	PCDOS NA	185
CB-86 - DRI	CPM86	600 439
Prof. BASIC Compiler	PCDOS	345 325
MACINTOSH COMPILER with BASICA syntax	MAC NA	325

FEATURES

- LINT-86 - finally a full lint to find subtle bugs - for all MSDOS Cs. \$200.
- PROLOG86 Interpreter for MSDOS includes tutorials, reference and good examples. Learn in first few hours. For Prototyping, Natural Language or AI. \$125.

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 FORTRAN PASCAL UNIX/PC or Debuggers, Linkers, etc.

EDITORS Programming

BRIEF - Intuitive, flexible	PCDOS NA	195
C Screen with source	8080/86 NA	75
FINAL WORD - for manuals	8080/86	300 215
MINCE - like EMACS	CPM, PCDOS	175 149
PMATE - powerful	CPM	195 175
	8086	225 195
VEDIT - full, liked	CPM, PCDOS	150 119
	8086	150 119

UNIX PC

COHERENT - for "C" users	PClike	\$500	475
COHERENT-NCI-Realtime	PClike	695	665
VENIX - "true V7" w/FTN	PClike	800	775
XENIX - "true S3" - rich	PC	1350	1295

Ask about run-times, applications, DOS compatibility, other alternatives. UNIX is a trademark of Bell Labs

LANGUAGE LIBRARIES

C Sharp Realtime-source	MSDOS NA	600
GRAPHICS: GraphiC-source	MSDOS NA	195
HALO - fast, full	PCDOS	200 165
Greenleaf for C - full	MSDOS NA	165
ISAM: C to dBASE-source	8086	150 140
CTree-Source, no royalties	ALL NA	375
dbVista-"Network," Source	MSDOS	495 465
BTRIEVE - many languages	MSDOS	245 215
Cindex + - no royalties	MSDOS NA	395
PHACT - with C	MSDOS NA	250
dBC - by Lattice	MSDOS NA	250
SCREEN: CView-validate	PCDOS NA	195
Databurst-C, BASIC	MSDOS	225 215
PANEL-86-many languages	PCDOS	295 265
WINDOWS for C-fast	PCDOS NA	139

RECENT DISCOVERIES

SCIL - Manage versions, changes to source code, documentation. Minimize confusion, disk space. Interactive. CPM 80, MSDOS \$349

FORTAN

ENVIRONMENT	PRICE	OUR PRICE
MS FORTRAN-86 - Impr.	MSDOS	\$350 \$ 225
Intel Fortran-86	IBM PC	NA 1400
DR Fortran-86 - full 77'	8086	500 349
PolyFORTRAN-XREF, Xtract	PCDOS NA	165

OTHER PRODUCTS

Assembler & Tools - DRI	8086	200 159
Atron Debugger for Lattice	PCDOS NA	695
CODESMITH-86 - debug	PCDOS	149 139
CURSES by Lattice	PCDOS NA	125
Disk Mechanic - rebuild	MSDOS	70 65
HS/FORTH - fast	PCDOS	220 210
IQ LISP - full 1000K RAM	PCDOS	175 call
MBP Cobol-86 - fast	8086	750 695
MicroPROLOG	PCDOS NA	285
Microsoft MASM-86	MSDOS	100 85
MSD Debugger	PCDOS	125 119
MultiLink-Multitasking	PCDOS	295 265
PFI-X-86 Debugger	MSDOS	195 169
PL/1-86	8086	750 495
PLINK-86 - overlays	8086	350 315
Polylibrarian - thorough	MSDOS	99 95
PolyMAKE	PCDOS	99 95
PROFILER-86 - easier	MSDOS NA	125
PROFILER - flexible	MSDOS NA	125
Prolog-86-Learn, Experiment	MSDOS NA	125
TLC LISP-86-full, liked	MSDOS NA	235
TRACE86 debugger ASM	MSDOS	125 115
X Shell-IF-THEN-ELSE	MSDOS	295 279

Note: All prices subject to change without notice. Mention this ad. Some prices are specials.

Ask about COD and PDs. All formats available.

0105

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

CIRCLE 69 ON READER SERVICE CARD

C LANGUAGE PROGRAMMING

From Plum Hall...the experts in C training

• C Programming Guidelines

Thomas Plum

• Learning to Program in C

Thomas Plum

Learning to Program in C 372 pp., 7 1/2" x 10", Price \$25.00

A practical, step-by-step guide for everyone acquainted with computers who wants to master this powerful "implementer's language" Inside, you will learn how to write portable programs for the full spectrum of processors, micro, mini and mainframe

C Programming Guidelines 140 pp., 7 1/2" x 10", Price \$25.00

A compilation of standards for consistent style and usage of C language. Arranged in manual page format for easy reference, it presents time-tested rules for program readability and portability.

PLUM HALL

1 Spruce Av, Cardiff NJ 08232

The experts in C and UNIX™ training. Phone orders: 609-927-3770

Please send me: _____ information on C and UNIX Training Seminars
 _____ copies of Learning to Program in C @ \$25.00/copy
 _____ copies of C Programming Guidelines @ \$25.00/copy
 NJ residents add 6% sales tax.

NAME _____
 COMPANY _____
 ADDRESS _____
 CITY/STATE/ZIP _____
 Check American Express Master Card Visa
 CARD # _____ EXP. DATE _____ / _____ Signature _____

UNIX is a trademark of AT&T Bell Laboratories

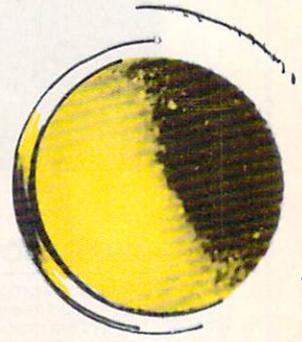
FREE C LANGUAGE POCKET GUIDE!

A handy C language programming pocket guide is yours free when you order either (or both) of the manuals above. A full 14 pages of valuable C language information!

CIRCLE 50 ON READER SERVICE CARD

erson

30 110 140 160



Exploratory Programming

By Michael Ham

Few computer languages offer a suitable environment for exploratory programming, which follows a winding path through idea and experience, hypothesis and experiment, and definition and test to arrive finally at solid and often serendipitous solutions.

To find the best solution, you must know the problem well. Such knowledge comes only from grappling with the problem and trying various holds. Once you truly understand the problem, through having tried a variety of approaches and solutions, you are able to step back, see the pattern of your experience with the problem, and watch the problem open to reveal its constituent sub-problems, from whose separate simple solutions the final program is quickly constructed.

Many of us are inclined to postpone the awkward first encounters with a new problem, a new situation in which we feel ignorant and vulnerable. Young children lack this fear: they are fascinated by something new and immediately set to work to find out what it can do—they hold it, taste it, shake it, listen to it, throw it, and in general develop a rich set of experience with it so that they can understand it. Somewhere along the way we abandon this method, productive as it is; we put our faith in analysis and acquire a fear of feeling ignorant.

Most computer languages pander to our disinclination to work directly with the problem. They encourage us to spend

time playing with paper—doing specifications, PERT charts, system designs, flow charts, and the like—and only at the very end actually to try to write the program code.

Logo and Forth, on the other hand, were designed to provide an environment that encourages exploratory programming. Logo's designers well understood the mechanisms of learning and were deliberately trying to mold the language to match the exploratory nature of children. Forth's designer was a practicing programmer who was deliberately trying to create a language to maximize his productivity in creating programs.

In Forth, for example, programmers normally begin the design process by writing and trying out various procedures, revising and refining them based on their effects and flaws until they begin to see through these prototypes what the problem truly requires. Only then do they attempt a final implementation, designed in the light of what they have learned. In Forth jargon this is called iterative development. How does Forth encourage this approach?

Exploratory programming is possible only under certain conditions. If the language implementation makes it difficult or tedious to define, compile, and test procedures, then the programmer will avoid writing short procedures because of the overhead effort. Putting more lines of code into a single procedure makes the overhead more tolerable.

Exploratory programming requires the define-compile-run cycle to be easy and quick, with minimal overhead, for only then will programmers write short procedures. It is only when the procedures

are short that they can be immediately and directly grasped by the mind. When the procedures are simple and elemental, they are easily combined with other such procedures to create more complex structures, which themselves are combined into bigger units that fit together ultimately to form the program itself.

Exploratory programming works best when procedures are named and called simply by name. In this way irrelevant mechanisms do not impede or interrupt the dialogue of thought and effect—the interplay between the idea as conceived and as realized. The name becomes (in our mind) the entity, and uttering the name invokes the action. We feel as if we are in immediate and direct contact with our thoughts as they take form in program code, and we watch them grow and evolve in a tiny Darwinian struggle with the problem.

The power of this exploratory approach can be seen in contexts other than programming, of course. Many of us have at some time or another helped in a mass mailing—a newsletter, for example. It obviously doesn't take high-level skills to design a work procedure to fold the letters, address the envelopes, insert the letters into the envelopes, seal the envelopes, and stamp them.

But if you've done this sort of thing, you inevitably find that soon one person is waiting while another struggles to keep up. The optimal solution is most quickly reached simply by setting to work,

observing the bottlenecks, and changing the procedures until things go smoothly. It usually takes three or four attempts before the procedure is completely polished, whereas a priori analysis and design would be more difficult, take longer, and probably produce a less efficient solution.

It is worth noting that the solution arrived at by experiment is optimal in part because it is not overgeneralized—it fits the specific requirements of *this* mailing and the skills and abilities of *these* people. It is important not to try to do too much; in programming as in art, less is often more.

Another example can be seen in forms design. No matter how careful the analysis and how considered the design, the first version of a new form is never right. The form inevitably fails to collect some needed information, a space is too small or too large for what people need to write in it, the layout is confusing, instructions are ambiguous, and so on.

Experienced forms designers quickly learn two lessons. First, when revising a form, they look at completed examples of the current version to discover how the form is actually filled out. Second, they print only a few copies of any new form, knowing the need for some revisions will immediately become obvious when the form is put to actual use. In other words, the designers build into their design cycle early and frequent interaction with the implementation of their ideas.

Yes-or-no Forth example

To look at an example of the exploratory approach in programming, let's examine the development of a new Forth command. Forth is an extensible language, and Forth programs are written by creating new commands and combining them to make new, higher-level commands until the final command is the program. Forth programmers find some commands generally useful in their applications, and they add these commands permanently to their Forth systems. The example given here is a command that would probably be wanted as a permanent addition.

A common exchange in interactive programs is for the program to ask a yes-or-no question and the user to respond. Thus it is convenient to have a yes-or-no com-

mand to collect and edit such a response. Our exploration will follow the evolution of a yes-or-no command in Forth.

The basic yes-or-no (*Y/N*) command is simply an indefinite loop that insists on getting a Y or an N and presents the result to the program:

```
: Y/N (-- n) 0 BEGIN DROP KEY
  DUP 78 = OVER 89 = OR UNTIL;
```

This works well enough. The phrase *0 BEGIN DROP* is a trick that gives a cheap way of setting up for repetitions of the loop. If the loop is repeated, *DROP* gets rid of any invalid input—input not equal to the ASCII values of N (78) or Y (89). The 0 preceding the *BEGIN* gives *DROP* something to work on the first time through the loop. An on-the-spot test (the word is compiled immediately upon the carriage return following the semicolon) shows that the word works as expected: it awaits a keystroke and leaves on the stack the ASCII value of the Y or N that was pressed, ignoring all other keys.

The definition can, however, be improved in several ways. First, the values 78 and 89 seem a bit stark by themselves. A reader of the program may not immediately realize their significance. And, of course, they depend on the base being decimal at the time the block is loaded from disk. Most Forths include the word *ASCII*, which accepts the next character and converts it to its ASCII value. The first enhancement is to replace 78 with *ASCII N* and 89 with *ASCII Y* to improve readability.

Often the output of *Y/N* is used by an *IF ELSE THEN* to pick the appropriate choice of action. In the preceding version of *Y/N*, however, both outputs, being non-zero, signify "true" to an *IF*. Thus the first functional refinement is to follow *UNTIL* with

```
ASCII Y =
```

This phrase makes the final output a true flag (if the input was Y) or a false flag (if the input was an N).

This change makes life easier for the programmer, but the focus of the programmer's attention should be the user. *Y/N* churlishly insists that the input be upper case. It will ignore a lower-case y

or n, even though either makes the user's intention perfectly clear. Programs should be written to respect the clear intention of the user.

One way to do this is to include two additional phrases in *Y/N*'s string of comparisons. These phrases would be placed between the *OR* and the *UNTIL*. The two phrases are:

```
OVER ASCII y = OR   OVER ASCII
n = OR
```

When you have a language such as Forth, however, in which you are defining new commands, a new measure of good design emerges. One must consider which functions should be embedded in the fabric of a given definition and which should be factored out for separate definition, with only their name included in the original definition. Generally speaking, a function that would make a good, elemental, general-purpose command should be given its own identity so that it can be used in any definition as needed.

In this case a better factoring is to isolate the function of capitalization to create a command that will capitalize any alphabetic input. *CAP* can be defined to do the job cheaply through a trick:

```
: CAP ( n -- n) 95 AND ;
```

You can insert *CAP* into the original definition immediately following *KEY*. Then a lower case y or n will be capitalized before the comparisons with the ASCII values are made. (You should think about why *CAP* works and what it does to non-alphabetic characters.) Now *Y/N* begins to resemble a usable word. What more can it do?

Without going into the details of development (these are left as an exercise for the reader), playing with the command leads to an evolution of refinement, with some subspecies created along the way.

For example, sometimes one wants the flags reversed so that a "no" response leaves a true flag and a "yes" a false flag. An example is the question "Do you want

to try again (Y/N)?” just before the *UNTIL* in a *BEGIN UNTIL* loop. (Of course, you could simply rephrase the question to “Do you want to stop now (Y/N)?” but never mind that.) A command *NO?* is created that accepts only yes or no answers but leaves reversed flags:

```
: NO? ( -- flag ) Y/N NOT ;
```

The definition of *NO?* is simple because we are able to use one of our earlier commands—already developed, tested, and at hand. This is an example of the advantage of an extensible language. Creating the foundation may require some pains, though it is made easier by tackling it in small chunks. As we work our way upward, the tools we command become more and more powerful.

Because *Y/N* will almost certainly be used after a question, its definition next is modified so that it displayed (once only!) the phrase “(Y/N)?” before attempting to collect the user’s response. We do this by including the phrase, “ (Y/N)?” in front of the *BEGIN*.

If the user will not be working in a situation in which noise is a problem, you might want to signal that an answer is sought by sounding a brief beep. It is easy to add a beep to the word by using your Forth command to sound a bell.

The beep could be made to sound once only (to notify the user an input is requested), even though several invalid inputs are received. Or the beep could sound at each request for input so that if several invalid inputs are received, the user will hear several beeps, one for each new input request. (The “(Y/N)?” display should, of course, occur only once.) In one case the beep is placed with the “(Y/N)?” before the loop, in the other just before *KEY* within the loop.

Sometimes it helps to show the user the response received so that she or he can tell that something has happened. *Y/N* is easily modified so that the response is displayed and then, if it is invalid, the cursor backs up and waits in the original position

for another attempt. (Note that the displayed phrase “(Y/N)?” must then be followed by a blank so the response is not jammed against the question mark.) A useful little word to echo input is easily defined:

```
: ECHO ( n -- n ) DUP EMIT ;
```

This command can follow *KEY* and will echo to the screen the character collected by *KEY*. The program must then backspace the cursor in the event of an error so that the new input will overwrite the old. At this point, the little trick of *O BEGIN DROP* reveals its inherent limitations: it inadequately separates the functions within the loop.

Y/N now has two distinct parts: the normal procedure and the setup for a repeat if the loop has to be retried. For this situation Forth provides the *BEGIN WHILE REPEAT* loop, in which the words between *WHILE* and *REPEAT* are executed only if the *WHILE* received a true flag (“execute the following while true”), and *REPEAT* always returns to *BEGIN*. If *WHILE* gets a false flag, the loop is escaped by execution proceeding immediately to the first word after *REPEAT*. Thus the *WHILE* clause functions naturally as the setup for a repeat of the loop. The definition can then be constructed (Listing 1).

Bells tend to annoy users—no one likes to be beeped at. The preceding word beeps only once, but it can be modified so that a beep occurs only when the input is invalid. If the user makes a valid response the first time, all is blessed silence. We simply move the *BEEP* from its position before *BEGIN* to within the *WHILE* clause—which in effect is before the *KEY* on attempts after the first one.

This latest version might be improved by displaying a message “Invalid

response” when the bell sounds for any invalid response. Note that the message must be blanked out when a valid response is received.

Because Forth, as a compiled language, is fast, it is probably simplest to include a *ZAP* word that prints 16 spaces (enough to wipe out the message) after the *ECHO*—the message can then be redisplayed by a word in the *WHILE* clause if the response is invalid. To the user’s eye, the message will barely flicker.

Sometimes a program can help the user by offering a default answer that the user can accept by entering a carriage return. A version of *Y/N* can be written to display a default *Y* and a version to display default *N*, each accepting a carriage return character as being the same as the default answer. (Obviously, if the default is a *Y*, for example, the user must still be able to enter a *Y* with no problem.)

Good factoring suggests that only one version of *Y/N* is needed to offer either default (*Y* or *N*). This is easily done by having *Y/N* expect a logical value that determines the default (for example, true = default *Y*, false = default *N*). The question using *Y/N* can then pass the appropriate value to *Y/N*. The new *Y/N* is defined as seen in Listing 2.

In this transfiguration, *Y/N* accepts a logical flag and then, after printing the question prompt, puts the ASCII value for *Y* or *N* on the stack (*Y* if the flag were true, *N* if false). That value is displayed to the user (with *ECHO*, note how our earlier extensions become generally useful), the cursor is backspaced to rest on the default answer, and *KEY* awaits the user’s input.

This time we don’t enter the indefinite loop until after we check to see whether the default answer was taken. If the keyed

```
: Y/N ( -- flag ) ." (Y/N)? " BEEP .BEGIN KEY ECHO CAP
  DUP ASCII N = OVER ASCII Y = OR NOT
  WHILE DROP BACKSPACE REPEAT ASCII Y = ;
```

Listing 1.

value was 13 (the ASCII for carriage return, factored out for separate definition as a documentation aid), then there is no need to go through the indefinite loop. The 13 is dropped (to expose the Y or N placed on the stack by the program), we ECHO the Y or N that the carriage return erased on the screen, and then we skip directly to the word that converts the Y or N to a logical flag.

On the other hand, if the user keyed some other value, we don't need our original Y or N. That value is swapped to the top of the stack and then dropped. We enter the indefinite loop, not to exit until a valid response is received. The loop itself has been factored out for separate definition because short definitions are easier to write and offer bugs fewer places to hide. Exploratory programming encourages us to write short definitions and compile and test them immediately—play with them to see what they do—and then use them with confidence.

VERIFY checks the value the user entered and, if it was invalid, continues to collect and edit responses until a valid response is received. Notice that KEY DUP has now migrated inside the WHILE clause (because in this version a KEY has already been done before the loop was entered, so within the loop KEY becomes

```

: CR? ( n -- flag ) 13 = ;
: VERIFY ( n -- n ) BEGIN ECHO CAP DUP ASCII N = OVER
      ASCII Y = OR NOT WHILE BEEP DROP BACKSPACE KEY REPEAT ;
: Y/N ( flag -- flag ) ." (Y/N)? " IF ASCII Y ELSE ASCII N THEN
      ECHO BACKSPACE KEY DUP CR? IF DROP ECHO ELSE SWAP DROP VERIFY
      THEN ASCII Y = ;

```

Listing 2.

PROGRAMMERS: New DAVID-DOS II™

HIGH-SPEED DOS 3.3

DOS-MOVER/FAST-GARBAGE

Makes high-speed disk access under all conditions. DOS-mover frees 10,000 bytes of extra memory for programs. FRE command does ultra-fast garbage collection. Clock or manual disk dating. May be licensed.

100 Sectors in 7 Seconds

Speed Load/Save Applesoft, Binary & Integer 100 sector programs in 7 seconds. Tload/Tsave Random & Sequential Text Files at same speed. Speeds up programs like Home Accountant.

10K More Memory

Use HIDOS command in hello program for turnkey startup, adding 10K free memory to run 30% larger Applesoft programs than ProDOS. Moves DAVID-DOS II and 4 buffers above main memory.

Ultra-Fast Garbage

New FRE command collects a memory full of 6000 strings that are half garbage in two seconds. DOS 3.3 takes 12 minutes. FRE is so fast it is not noticeable during run of most programs.

Clock Dating

Automatic date stamping of disk files is set up for 6 kinds of clocks or Hello manual date entry. New DATE command will auto-insert date in correspondence.

All times in seconds. (Time Test programs available)		DAVID DOS-II	ProDOS	DOS 3.3
TEXTFILES (100 Sectors) (791 Strings, 32 chars ea) (442 Sectors, 7 x 500)	TSAVE	8.0	NO	NO
	TLOAD	6.2	NO	NO
	WRITE	29.3	28.0	88.4
	READ	24.3	16.3	83.8
	PRINT/READ	44.2	45.9	117.1
APPEND	142.3	142.9	1231.2	
APPLESOFT (100 Sectors)	*SAVE	6.4	16.4	33.1
	LOAD	5.0	4.0	23.5
INTEGER (100 Sectors)	*SAVE	6.6	NO	33.4
	LOAD	4.9	NO	23.4
BINARY (100 Sectors)	*BSAVE	7.3	18.4	28.7
	BLOAD	5.8	4.8	24.5
48K PROGRAM SPACE (With 3 Bufs avail)	APPLESOFT	36.352	NO	36.352
	INTEGER	36.352	NO	36.352
	BINARY	36.352	34.816	36.352
64K PROGRAM SPACE (With 4 Bufs avail)	APPLESOFT	46.592	32.256	35.756
	INTEGER	46.592	NO	35.756
	BINARY	46.592	41.728	35.756

*Add 5 seconds for Verify. Apple II, Applesoft & ProDOS are trademarks of Apple

Ten New DOS Commands

1. HIDOS moves DOS above 48K memory.
2. FRE makes ultra-fast garbage collection.
3. DATE stamp files. Clock or manual dating.
4. TLOAD speed loads all Text Files to mem.
5. TSAVE speed saves Text Files from memory.
6. TLIST lists all Text Files to screen/printer.
7. DUMP Binary/Ascii to screen or printer.
8. DISA disassembles Binary to screen/printer.
9. AL prints program Address & Length.
10. / is a one keystroke Catalog.

DAVID DATA™

CIRCLE 17 ON READER SERVICE CARD

Identical Operation

All new commands operate identical to old DOS commands on the keyboard & in programs.

Install in Three Seconds

Install DAVID-DOS II on your full disks in three seconds without touching the programs. Create bootable high-speed new disks with a Basic, Binary, or Exec Hello & 35/40 tracks. Create Data-Disks with 30 extra sectors.

Variable Speed Scrolling

Key operated variable speed scrolling for TLIST, DUMP and DISA. Lower-case accepted on all commands. Catalog shows Free-Space. Automatic support of Integer or Applesoft Card in any slot, while in HIDOS or LODOS. Vendor license includes protection system.

Compatible

All DOS entry addresses have been preserved. DOS is same length and compatible with most software. David-Dos is fully copyable. Init areas were used for David-Dos. Works with all Apple IIs including Iie, Iic, Franklin & Basis with 48K/64K/128K & Corvus & Xebec Hard Disks. Complete documentation and many utilities are on the disk.

Add \$2.00 Shipping. Overseas add U.S. \$4.00. Calif. add 6%. **\$39.95**

To Order: Send Check or Phone Visa/MasterCard.
12021 WILSHIRE BLVD., SUITE 212E
LOS ANGELES, CA 90025 (213) 478-7865

a part of the setup for a repetition rather than the normal routine). Note also that in this version the beep is sounded only for invalid input.

Playing with this most recent definition shows that the *CAP* and *ECHO* in *VERIFY* should be interchanged so the user will see a capitalized Y or N (as is shown for the default) even if lowercase were entered. But in that case numeric entries give odd responses until *CAP* is redefined to replace its little trick with a more explicit test. (Note how tricks often fail to hold up as you play and experiment with them; cheap tricks, like cheap toys, are frequently frail.) The new definition for *CAP* is seen in Listing 3.

The user, with this latest definition of Y/N, cannot accept the default value after once attempting some other answer. This is a matter of the programmer's choice, of course, but since we elected to display the invalid response and thus overwrite the default answer, it seems best that the default answer should vanish internally as well so that the program will be consistent with the screen display. It is good practice to match the user's mental model of what is happening. If the value is displaced on the screen, it should no longer exist in the program.

An even better default situation would be for the question to remember the last answer it got and pass that value to Y/N as the default value. Each question could have its own variable in which the last answer is stored. The question passes its variable's address to Y/N, which fetches the value that determines the default and then, after completing its input, stores a copy of the new answer for the next time the question is asked.

A little experience shows that Y/N should be written so that one value of the variable indicates no default is presumed. This value would be the variable's initial value for the first time the question is asked.

The article ends here, but the evolution of the word can continue. Take a moment to reflect, however, that it is unlikely the

programmer would have arrived at this last embodiment of Y/N if he or she had taken the initial specification and immediately designed the word, without playing with the idea and letting it develop and grow.

Exploratory programming allows the programmer to have much experience within the writing of a single program. Hindsight is one of our most powerful tools; exploratory programming allows you to use it early and often. ■

Michael Ham has worked in program and systems design, development, and documentation for many years. He currently works in software development at the Scotts Valley, Calif., office of Dysan Corp.

```
: CAP ( n -- n ) DUP ASCII a < NOT OVER ASCII z > NOT AND
IF 32 - THEN ;
```

Listing 3.

(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: LISPTX 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 280 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

Scroll & Recall™

*Screen and Keyboard Enhancement
for the IBM - PC, XT and Compatibles*

*Allows you to conveniently scroll
back through data that has gone off
the top of your display screen.*

*Allows you to easily recall and edit
your previously entered DOS com-
mands and data lines.*

*Very easy to use, fully documented.
Compatible with all versions of DOS,
monochrome & graphic displays.*

\$69 - Visa, M/C, Check, COD, POs
Phone orders accepted

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 Inquiries Welcome

CIRCLE 66 ON READER SERVICE CARD

Six Times Faster!

Super Fast Z80 Assembly Language Development Package

Z8OASM

- Complete Zilog Mnemonic set
- Full Macro facility
- Plain English error messages
- One or two pass operation
- Over 6000 lines/minute
- Supports nested INCLUDE files
- Allows external bytes, words, and expressions (EXT1 * EXT2)
- Labels significant to 16 characters even on externals (SLR Format Only)
- Integral cross-reference
- Upper/lower case optionally significant
- Conditional assembly
- Assemble code for execution at another address (PHASE & DEPHASE)
- Generates COM, HEX, or REL files
- COM files may start at other than 100H
- REL files may be in Microsoft format or SLR format
- Separate PROG, DATA & COMMON address spaces
- Accepts symbol definitions from the console
- Flexible listing facility includes TIME and DATE in listing (CP/M Plus Only)

SLRNK

- Links any combination of SLR format and Microsoft format REL files
- One or two pass operation allows output files up to 64K
- Generates HEX or COM files
- User may specify PROG, DATA, and COMMON loading addresses
- COM may start at other than 100H
- HEX files do not fill empty address space.
- Generate inter-module cross-reference and load map
- Save symbol table to disk in REL format for use in overlay generation
- Declare entry points from console
- The FASTEST Microsoft Compatible Linker available

SPEED!
SPEED!
SPEED!

- Complete Package Includes: Z8OASM, SLRNK, SLRIB - Librarian and Manual for just \$199.99. Manual only, \$30.
- Most formats available for Z80 CP/M, CDOS, & TURBODOS
- Terms: add \$3 shipping US, others \$7. PA add 6% sales tax

For more information or to order, call:

1-800-833-3061

In PA, (412) 282-0864

Or write: SLR SYSTEMS

1622 North Main Street, Butler, Pennsylvania 16001

CIRCLE 59 ON READER SERVICE CARD

SLR Systems

U N I X S O F T W A R E

**UniPress
Product
UPDATE**

LATTICE® C NATIVE AND CROSS COMPILERS FOR THE 8086

AMSTERDAM COMPILER KIT

Outstanding software development tools

Lattice C Cross Compiler to the IBM-PC

- Highly regarded compiler producing fastest and tightest code for the 8086 family.
- Use your VAX or other UNIX machine to create standard Intel object code for your 8086 (IBM-PC)
- 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 included.
- Send and Receive communication package optionally available to communicate between Unix and MS-DOS.

Hosted On

Prices: VAX/Unix and VMS	\$5000
MC68000/8086	3000
Send and Receive	500

Lattice C Native Compiler for the 8086

- Runs on the IBM-PC under MS-DOS 1.0 or 2.0.
 - Produces highly optimized code
 - Small, medium, compact and large address models available.
 - Compiler is running on thousands of 8086 systems.
- Price: \$425

Plink (Optional) for use with native Lattice

- Full function linkage editor including overlay support.
- Price: \$395

Amsterdam Compiler Kit

- Package of compilers, cross compilers and assemblers.
 - Full C and Pascal language.
 - Generates code for VAX, PDP-11, MC68000, 8086 and NSC16000.
 - Hosted on many Unix machines.
 - Extensive optimization.
- Price: Full system—source \$9950
Educational Institution 995

OEM terms available • Much more Unix software, too! • Call or write for more information.

Mastercard and Visa

UniPress Software, Inc.

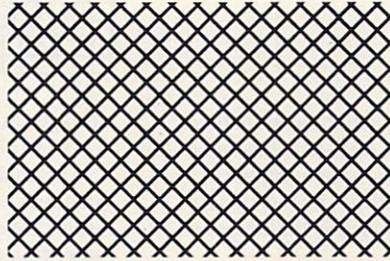
2025 Lincoln Highway, Edison, NJ 08817
201-985-8000 • Order Desk: 800-222-0550 (outside NJ) • Telex 709418

Lattice is a registered trademark of Lattice, Inc. Unix is a trademark of Bell Laboratories. MS-DOS is a trademark of Microsoft.

U N I X S O F T W A R E

Fred

More than just a macro facility
within Framework



By Darryl Rubin



What would you call a language that combines some of the best features of LISP, Smalltalk, BASIC, and C? The developers of just such a language at Forefront Corp. have chosen to call it Fred.

Fred is the programming language component of Framework, the integrated software package distributed by Ashton-Tate for the IBM PC. But Fred is much more than the mere macro facility you might expect from a package rooted in the world of multifunction spreadsheets.

Despite its unassuming name, Fred is a full-featured programming language that lets you manipulate text, numbers, and graphics on the same screen in exciting new ways. Fred puts all of Framework and its flashy multi-windowed environment at your command. The only trick in using it is divining what is missing from its skimpy documentation.

Well sit back and discover.

In this article we'll tour the Fred language, learn exactly how to write and run Fred programs, and develop some useful routines that will make Framework work better for you. We'll even trip across one of Fred's useful but undocumented features.

You can get your own copy of the programs presented in this article by calling the *COMPUTER LANGUAGE* Bulletin Board Service at (415) 957-9370 and downloading the file FRED.FW. If you don't have Framework, download the file FRED.LST instead.

The anatomy of Fred

In the true spirit of integration, Fred pulls together features from several of today's most influential programming languages.

Like Smalltalk, Fred operates in a multi-windowed environment that encapsulates programs and data in displayable objects—frames—that can communicate with each other and be nested hierarchically. For example, you can send data to, display, and then hide a frame as easily as this:

```
@SET(FRAME1,DATA),  
@DISPLAY(FRAME1),  
@HIDE(FRAME1).
```

Like LISP, Fred has a recursive, function-oriented syntax with flexible type binding and the ability to pass a variable number of arguments to a function. Programs are stored and treated like data, so programs can modify and create other programs. Do you recognize this famous function as written in Fred?

```
F(n): @if( @or(n=0,n=1), 1, @F(n-1))
```

Like C, Fred is a free-formatted, block-structured language in which spaces, tabs, and comments are insignificant, as are—believe it or not—character fonts and enhancements (yes, you can italicize and embolden your source text!). Fred's other C-like features are its local and external variables, *if/then/else* and *while* statements, and value-returning assignment operator.

There's even a clever way to write *#DEFINE* macros in Fred, as we'll see. A C programmer should feel right at home with the following Fred statement:

```
@while ;Open a while block  
( ;Condition clause:  
(c := @get(Table)) <> #N/A!,  
; statement:  
total := total + c,  
)
```

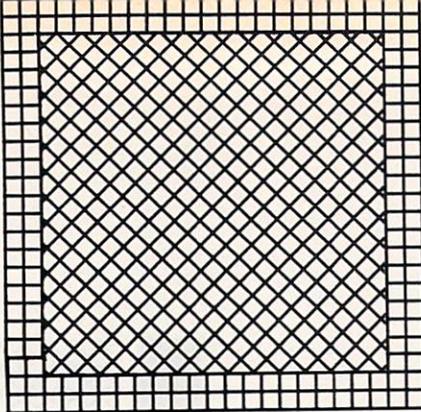
Like BASIC, Fred is an interpretive language with interactive editing and diagnostics features. When an error occurs, Fred shows you the location of the error in the source program and tells you in plain English what went wrong. A *@TRACE* function acts like a supercharged version of the *TRON* and *TROFF* statements of many BASICs. With this and Framework's full-powered word processing at your disposal, you can usually correct and rerun your program in seconds.

Like any full language, Fred has an extensive set of data types, operators, and built-in functions. This is the meat-and-potatoes part of the language, and Fred will leave you, if anything, stuffed.

Framework's manual tells you about Fred's numeric and string data types, but you'll have to read very, very carefully to realize that Fred has arrays and structures, too. (Figuring this out qualifies you to join Mensa.)

Operators include everything you'd expect plus some you don't always get: exponentiator (\wedge), string concatenator (&), range operator (:), and a unique postfix operator, percent (%).

The set of built-in functions defies



description, other than to say it includes functions for math, statistics, business, string manipulation, logic, graphics, sound, print formatting, menus, macros, program control, function building, and frame navigation.

Surely, you say, a language of so many parts must be the Edsel of the eighties, the PL/I of the PC generation. Not at all. Fred is surprisingly clean and straightforward. I would almost call it elegant because many of its features are rooted in a single, underlying construct.

And that is the frame.

What's in a frame

The frame is the unifying construct of Framework. It is at once the basic unit of

storage, execution, and display within the Framework environment.

This makes frames much like objects in Smalltalk. Frames can hold data and programs, be executed, communicate with other frames, and display themselves in overlapping windows. Frames can also be nested hierarchically, inheriting programs and data from their containing, parent frames. This gives Fred a powerful block structuring capability.

A frame is like a two-sided sheet of paper. Side one is the value area that holds whatever data is associated with the frame. This can be a number, string, graphics, spreadsheet tables, data base records, or other (nested) frames.

Side two of the frame is the formula area. It holds the program by which the value part of the frame is calculated or processed. Framework normally doesn't

display this side of a frame, but it is where Fred lives and works.

Any frame can have a formula and be nested inside other frames. You can also associate formulas with spreadsheet cells and data base fields because Framework treats these as frames, too—nested frames to be exact.

Figure 1 shows a Framework desktop with text, spreadsheet, data base, and nested frames. What you're seeing here is the value area of each of the frames. Notice that each frame has a label in its top-left corner. As we'll see later, this is the name by which a Fred program can reference the frame's value area or execute (call) the frame's formula.

The frames labeled UTILS (OUTLINE VIEW) and FRAME VIEW in Figure 1 are examples of frames that contain other frames. In fact, these are two different views of the very same frame. Where UTILS displays its nested frames in outline form (labels only), FRAME VIEW shows the frames themselves. Look carefully—you're getting a sneak preview of the programs we'll be presenting shortly.

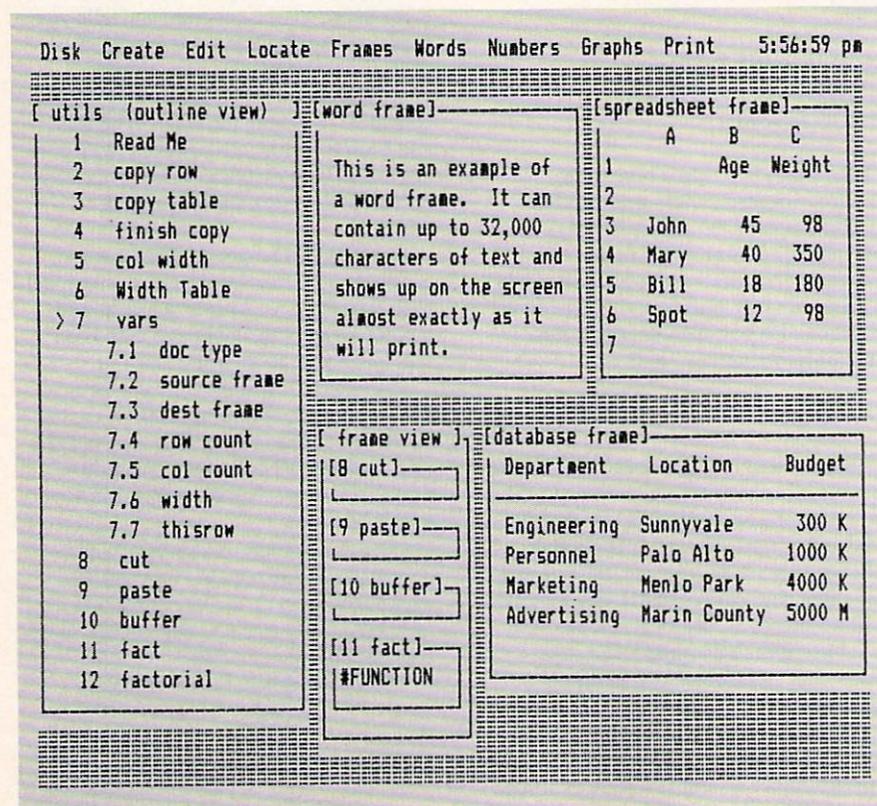


Figure 1.

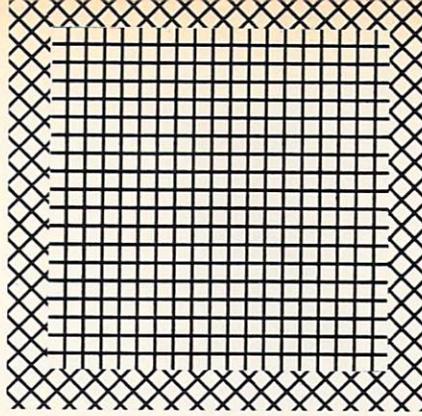
Freedom of expression

A Fred program probably resembles a LISP program more than anything else because it is simply an expression or a sequence of expressions. As such, a Fred program always returns a value, which is the value of the last evaluated expression. For example, here is a program called Factorial whose value is the factorial of a number entered by the user:

```
@prompt(@integer(@fact(
  @value(@inputline(
    "Type a number")))))
```

To decipher this program you have to read it inside out because that is the order in which the functions are evaluated. The @INPUTLINE function prompts the user and returns keyboard input as a string. @VALUE converts this to a number and @FACT evaluates the factorial of that number. @INTEGER converts the result to a string and @PROMPT displays it.

@INPUTLINE is one of Fred's snazziest functions because it lets the user zoom into a full-screen editing mode if he or she needs to submit more than one line of text. All of Framework's word processing fea-



tures are then available for editing the input, including text formatting and display enhancements.

The preceding `@FACT` function is one that I wrote. It does the dirty work of calculating factorials in true recursive style:

```
@if( ;condition:
  @item1 = 1, @item1 = 0),
  1, ;then
  @item1 * @fact(@item1-1)) ;else
```

These lines illustrate Fred's `@IF` statement, which is simply a list of three expressions: a condition, a *then* (true) clause, and an *else* (false) clause. Sounds mighty like the LISP `COND` function, doesn't it?

Each clause of the `@IF` statement can only be a single expression. If one just won't do, you can turn a list of expressions into a single one by using the form `@LIST(EXPR1, EXPR2, . . . , EXPRN)`. Like Lisp's `PROGN` function, `@LIST` sequentially evaluates each of its arguments and returns the value of the last one.

Notice `@FACT`'s use of the `@ITEM1` function, which returns the value of the first argument passed to a Fred program. Other arguments, if supplied, could be referenced using `@ITEM2` through `@ITEM16` or by the expression `@ITEM(NUMBER)`. You can use `@ITEMCOUNT` to find out how many arguments were actually passed.

See Fred run

Fine, you say, but how can you enter and execute programs like `@FACTORIAL` and `@FACT`? The Framework documentation probably left you a little bewildered. Here's all there is to it:

1. Create a frame of the desired type or highlight the border of an existing one.
2. Press F2 (Formula Edit) followed by F9 (Zoom).
3. Enter your program. All of Framework's word processing features are available to you.

4. When you're done, hit escape and label your frame if you haven't already done so. The frame's label serves as the name for your program.

Because Fred treats spreadsheet cells and data base fields like frames, you can associate programs with them, too. Just highlight the desired cell or field and proceed from step 2. You won't have to individually label the cell or field because cells are named by their coordinates (e.g., B3 or, equivalently, Age.John in Figure 1), while fields are named by the first row of the data base (e.g., Budget in Figure 1).

Framework offers you several ways to run your program. One is to highlight the cell, field, or frame containing the program and press F5 (Recalculate). Another is to bind the program to an Alt key. To do this you'll need to put a call to `@SETMACRO` into an empty frame and execute it via F5. For example, executing `@SETMACRO({ALT-F}, UTILS.FACTORIAL)` will bind the factorial program to Alt-F.

The third way to execute a program is to call it as a procedure from another program. Look back at the `@FACTORIAL` program presented in the last section. Did you notice how this program called `@FACT`?

You might be wondering why procedure calls require that you preface the called frame's name with an at sign (`@`). For the answer, let us proceed to the reference section.

Fred's references

The C language lets you refer to programs and data using symbolic identifiers. Fred has identifiers too, but in Fred they are called references.

Using references, you can execute another frame's formula, retrieve or change that frame's value, or perform an operation on a range of adjacent frames. Corresponding to each of these respective purposes, Fred has formula references, value references, and region references.

A formula reference is the name of a frame preceded by an at sign (`@`) and optionally followed by a parameter list in parentheses. This is just like a function call in C. For example, `@FACT(2)` calls `@FACT` to evaluate the factorial of 2. More in the LISP style, parameters are

passed by value and can be of any type. You can even pass different numbers of parameters on each call to the same frame.

A value reference is just the name of a frame. Like an identifier in C, a value reference can be the source or target of an assignment or a term in an expression. For example, the UTILS frame in Figure 1 has within it a frame called BUFFER. Valid references to this frame's value include `BUFFER := "THIS IS A STRING"`, `BUFFER := 2`, `BUFFER := BUFFER + 1`, and `BUFFER := VARS.DOC TYPE`. You see, the value area of a frame is just like a C external variable except it is untyped.

A region reference is a pair of value references separated by a colon. For example, `UTILS.CUT:UTILS.BUFFER` refers to the frames UTILS.CUT, UTILS.PASTE, and UTILS.BUFFER in Figure 1. Similarly, `C3:C6` and `WEIGHT.JOHN:WEIGHT.SPOT` each refer to the weight column of the spreadsheet frame.

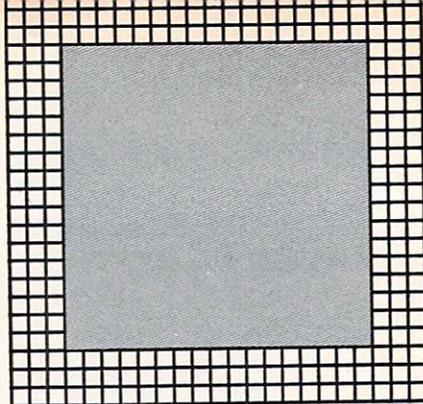
Given Framework's ability to have multiple frames on the desktop and to nest frames, you might be wondering what happens if you have two frames, say X and Y, each of which contains a frame called Z. No problem, because references can use a path-naming syntax, as in `X.Z` and `Y.Z`. In fact, when a program in one frame refers to another frame, it must use the full path name of the target frame unless that frame is the parent (containing frame) or a brother of the referencing frame.

Some examples of path names in Figure 1 are `UTILS.VARS.WIDTH`, `SPREADSHEET FRAME.B3`, `SPREADSHEET FRAME.AGE.JOHN`, and `DATABASE FRAME.BUDGET`.

Kinder cuts

Now that we're getting familiar with Fred, let's put the language through its paces.

In its utility library, Framework includes two programs, `CUT` and `PASTE`, that let you cut text from a word frame and later paste it back. But these programs are `s . . . l . . . o . . . w` and can only handle one cut at a time.



Listing 1 shows revved up versions of CUT and PASTE that not only work almost instantly but also let you stack multiple cuts and recall them in reverse order.

These programs introduce a few of Fred's most useful functions: @TEXTSELECTION, @PERFORMKEYS, and @SETSELECTION.

@TEXTSELECTION returns as a string the currently highlighted text in the currently selected frame; this is how CUT fetches the text to be cut.

@PERFORMKEYS is a keyboard function that passes Framework a string of keystrokes as if the user had typed them.

CUT uses this to activate the move function (F7) in preparation for moving the highlighted text to the cut buffer (UTILS.BUFFER).

@SETSELECTION navigates to the specified frame and returns the name of the currently selected frame as a string. CUT uses this function to navigate from the user's word frame to UTILS.BUFFER, which is the destination of the move operation.

Also notice the @LOCAL function. This function declares a local variable for use by a frame's formula. Local variables can hold any value that a frame can, even a text string up to 32,000 characters long. In fact, the name of a local variable can be used in value and formula references just like frame names can.

You can do interesting things with this capability, as we'll see next.

Can do

You've already seen Fred's @IF and @WHILE statements, but what about the good old "do" loop? Fred doesn't have one, but we can write one, as Listing 2 proves.

DO's most interesting feature is how it assigns its first parameter (a text string) to a local variable (REF) and executes that string as a formula (via @REF).

This is a powerful capability. As in LISP, your programs can synthesize and execute other programs. You can also use this feature to implement C-like #DEFINE macros. For example, the following macro substitutes "This is a bother to type" wherever you type @ABBREV:

```
abbrev := "@list('"'This is a bother to type'"")"
```

A more useful macro is @DIALOG:

```
dialog := "@erasesprompt," &
"@prompt('"'Enter your choice'"')," &
"k:=@nextkey"
```

You can assign formulas to frames as well as to local variables. Fred provides a function called @SETFORMULA for this purpose. There is also a function called @GETFORMULA, which returns a frame's formula as a string that you can manipulate.

Frame work

Now let's really put Fred to work. Have you ever tried to copy a table from a spreadsheet or data base to a word frame? If so, you've seen Framework politely tell you that it can't be done. Well it can be done if you're familiar with Fred, as the @COPY TABLE program in Listing 3 demonstrates.

This program actually consists of several subroutine and data storage frames, numbered 2 through 7.7 in the UTILS frame (Figure 1). The whole thing is too long to list or explain here, so let's just hit the highlights.

@COPY TABLE, the main routine, is invoked when you hit Alt-T. It prompts



```
UTILS.DO:
```

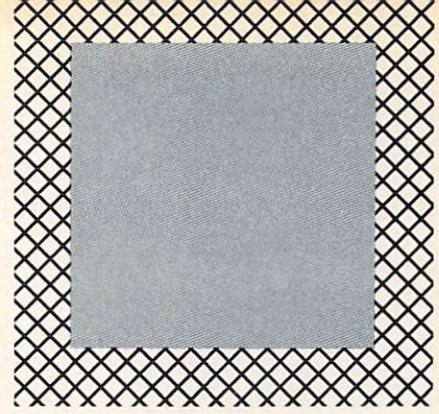
```
; Do -- Repeated do loop.
; Syntax: @do(expression,count)
;
; Result: Executes the passed expression "count" times.
;
; Note: The expression must be passed as a string value. For
; example, to sound 3 beeps: @do("@beep(880,100)",3)
```

```
@local(i,ref),
ref := @item1,
i := @item2,
@while(i > 0,
@ref,
i := i - 1)
```

```
UTILS.DOLOOP:
```

```
; Doloop -- demo driver for Do routine. Prompts user for a Fred
; expression and a count, then calls Do execute the
; expression the specified number of times.
@do(@inputline("Type a Fred expression"),
@value(@inputline("Type a repeat count")))
```

Listing 1.



you for information about the table to be copied and the name of the frame you want it copied to. It then makes multiple calls to `@COPY ROW`, which does the dirty work.

The secret of `@COPY ROW` is its use of the `@TEXTSELECTION` function to retrieve the contents of spreadsheet cells and data base fields as screen formatted strings. Wait, you say, the Framework documentation says that `@TEXT SELECTION` only works in word frames. Surprise—here is one of Fred's undocumented and quite useful features. `@COPY TABLE` shows off some other

nifty features of Fred. For example, it lets you specify whether the table should have fixed or varying column widths. If the latter, it presents you with a data base frame, `WIDTH TABLE`, into which you enter the desired column widths. Constructing a fancy user interface was never so easy.

Another neat thing `@COPY TABLE` does is use `@SETFORMULA` to create a function, `@COL WIDTH`, that returns the width of the current column to `@COPY ROW`. This way, `@COPY ROW` doesn't have to know whether you specified fixed or varying column widths. If you specified a fixed width, `@COL WIDTH` is assigned a formula that returns a constant value. Otherwise it gets a formula that looks up values in the `WIDTH TABLE`.

Does it strike you that the `WIDTH TABLE` serves as an array for `@COL WIDTH`? Indeed it does—you've found Fred's "missing" array and structure construct: the data base frame. (Welcome to Mensa!) If you like, you can also use spreadsheet frames to store arrays and structures.

Frame play

After all the features we've seen so far, could Fred possibly have more to offer? The answer is a resounding yes! Fred also

```
UTILS.CUT:
```

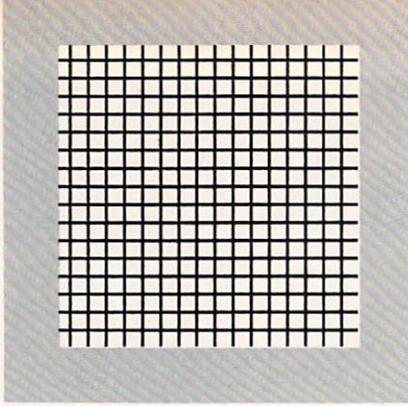
```
; Cut -- Cut text to a cut buffer. Uses the global variable "buffer".
```

```
@echo(#OFF), ;Keep the display quiet
@local(src),
@performkeys("{f7}"), ;Start the move sequence
src := @setselection("utils.buffer"), ;Go to the cut buffer
@performkeys( ;Finish the move sequence,
"{dnlevel}{ctrl-home}{return}{ctrl-home}{return}"), ; stacking the moved
; text LIFO
@setselection(src), ;Return to the source frame
@performkeys("{dnlevel}"),
@echo(#ON) ;Reenable display
```

```
UTILS.PASTE:
```

```
;Paste -- Paste text from cut buffer. The last stacked entry in the
; buffer is the one pasted.
```

```
@echo(#OFF), ;Keep display quiet
@local(dest),
dest := @setselection("utils.buffer"), ;Go to the cut buffer
@performkeys( ;Select the topmost buffer
"{dnlevel}{ctrl-home}{ctrl-pgdn}{f7}"), ; entry and start the move
@setselection(dest), ;Go to the dest frame
@performkeys("{dnlevel}{return}{backspace}"), ;Finish the move and
; delete the terminating
@echo(#ON) ; paragraph marker
```



does menus, graphics, print formatting, DOS environment control, and external program calling.

With all these capabilities at your command, you can do more than make frames work—you can positively make them sing. I'm serious, because Framework comes with a sample program, MUSIC-

MAC, that lets you write and play music.

Considering its ability to manipulate text, numbers, and graphics on the same screen and its combination of features from Smalltalk, LISP, C, and BASIC, will Fred make obsolete other languages? Is it the programmer's be all and end all? Is it the only programming language you'll ever need?

Probably not. Fred doesn't have everything for everybody.

It just has a little bit of something for everybody. ■

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

UTILS.COPY TABLE:

```
;Copy Table -- Copies a selected table from a spreadsheet
;                or database to a word frame.

;Local variables --
; rows:  Number of rows to copy
; cols:  Number of columns to copy
; src:   Source frame (spreadsheet or database)
; dest:  Dest frame (word frame)

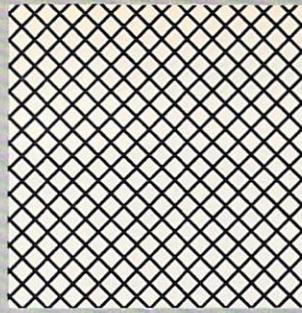
;Global variables (contained in Vars frame) --
; doc type: Source frame type (s => spreadsheet, d => database)
; width:    Column width of table or "p" to flag use of per-column widths
; thisrow:  Number of current row being processed
; row count: Total number of rows to copy
; col count: Total number of columns to copy

;User defined functions --
; @Copy Row(cols,dest frame): Copies the specified number of columns
;                             to the specified destination frame.
;
;
; @Col Width(column number): A function created by Copy Table based
;                             on whether the user specified a fixed
;                             column width or per-column widths.
;                             @Col Width is used by @Copy Row.
;

@local(rows,cols,src,dest),
vars.doc type := "",
vars.thisrow := 0,                                     ;current row being copied

@while(                                               ;ask whether source frame
  @and(vars.doc type <> "s",                          ; is spreadsheet
        vars.doc type <> "d"),                        ; or database
  vars.doc type := @inputline(                        ; and store user response
```

Listing 3. (Continued on following page).



```
"Is the source frame a spreadsheet or database?" &
" (type s or d))),

rows := @value(@inputline(                ;ask for number of rows
"Enter number of rows to copy")),        ; to copy

cols := @value(@inputline(                ;ask for number of columns
"Enter number of columns to copy")),      ; to copy

dest := @inputline(                       ;ask for name of dest
"Enter name of word frame to copy table to"), ; word frame

vars.width := @inputline(                 ;ask for column width
"Enter column width or type 'p' to specify per-column widths"),

                                           ;initialize @col width
                                           ; function, assuming
                                           ; user picked a fixed width
                                           ;Return fixed column width
@setformula(utils.col width,
             @value(vars.width)),

@if(@iserr(@value(vars.width)),          ;but if user didn't type
                                         ; a number, he wants per-
                                         ; column widths, so
@list(@performkeys("{uplevel}"),        ; copy local vars to globals
      utils.vars.source frame := @setselection(), ; for use by @finish copy
      utils.vars.dest frame := dest,      ; routine
      utils.vars.row count := rows,
      utils.vars.col count := cols,

                                           ;then set @col width function
                                           ; to use per-column width

@setformula(utils.col width,
             @list(@local(w,i),
                   i := @iteml,           ; get passed column number
                   w := @get(utils.Width Table.Width), ; get width of 1st column
                   @while(i>0,@list(     ; step through Width Table
                     i := i-1,          ; to get width of
                     w := @next(utils.Width Table.Width))), ; desired column
                   w)                  ; and return that width
             ),

@setselection("utils.Width Table.width"), ;all set, now take user
@performkeys("{f9}"),                   ; to Width Table
@eraseprompt,                            ; and let him edit column
@prompt("Edit width values, press Alt-U when done")), ;width values
```

Listing 3. (Continued on following page).

```

;If user is specifying per-column widths, this function now returns.
;The user must edit the width values in the Width Table and then press
;Alt-U to invoke @finish copy, which will complete the copy process.
;If the user specified a single column width for the whole table,
;processing continues with the @list block below.

@list(
  @while(rows > 0,@list(                                ;do for each row
    @copy row(cols,dest),                               ;copy this row to dest
    @performkeys("{dnarrow}"),                         ;advance to next row
    src := @setselection(dest),                        ;go to dest frame
    @performkeys("{dnlevel}{end}{return}"),           ;and insert a line break
    @setselection(src),                                ;return to source frame
    vars.thisrow := vars.thisrow + 1,                 ;bump row counter

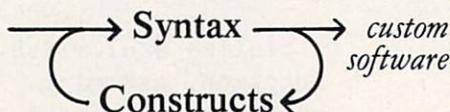
    @if(vars.doc type = "d",@performkeys(              ;if source is a database,
      @rept("{dnarrow}",vars.thisrow))),              ; return to current record

    rows := rows-1)),                                  ;downcount row processed

  @setselection(dest),                                 ;go to dest word frame
  @performkeys("{dnlevel}"))                          ;leave user at end of
                                                    ;table copied

```

Listing 3. (Continued from preceding page).



Get the power of your Z80
and the elegance of direct access
to CP/M functions from your
high level programs with

SYNLIB

utility library

SYNLIB consists of MICROSOFT compatible object code that may be called from any high level language that uses MICROSOFT parameter passing conventions.

SYNLIB gives you extremely powerful array and buffer manipulation using the Z80 LDIR instruction; program access to the CP/M CCP console command line; high speed disk block I/O; a high quality random number generator; HEX to ASCII conversion optimized by special Z80 instructions; program chaining and more.

And, because our programmer abhors a vacuum, each 8" floppy comes packed with MODEM7 and other valuable public domain software. We've included documentation and available source, so that you too may join the free software movement.

SYNLIB \$50.00

8" SSSD CP/M format

SOURCE: \$100.00

Licensing for commercial use available.

SYNTAX CONSTRUCTS, INC.

14522 Hiram Clarke
Houston, Texas 77045
(713) 434-2098

CP/M is a registered trademark of Digital Research, Inc. Microsoft is a registered trademark of Microsoft Corp. Z80 is a registered trademark of Zilog.

CIRCLE 73 ON READER SERVICE CARD

OPT-TECH SORT™

SORT/MERGE program for IBM-PC & XT

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
- Handles variable and fixed length records
- Supports all common data types
- Filesize limited only by your disk space
- Dynamically allocates memory and work files
- 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, includes on-line help feature
- Full documentation — sized like your PC manuals
- **\$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 67 ON READER SERVICE CARD

Enhancing Source Code Control under UNIX

PART II

By Luke C. Dion and Alan Filipski

Last month we briefly described the Source Code Control System (SCCS) utilities *admin*, *get*, and *delta* provided with the UNIX operating system and described how a simple front-end interface could greatly enhance the usefulness of these utilities for a software project. This article provides some technical detail on how to actually implement such an interface.

The implementation described here, the Project Source Code Control System (PSCCS), was used by Motorola Microsystems Inc. during a UNIX System V port involving about 6,000 files and 15 people.

Such an interface is not difficult to build; ours consisted of only 600 lines of C code and was tailored to the requirements of our project. The information presented here should enable a programmer to write such a package for any particular project environment with any enhancements required.

Anyone doing this should, of course, become thoroughly familiar with the UNIX documentation on the utilities *admin*, *get*, and *delta*, as well as the less frequently used *prs*, *cdc* and *rmdel*. Part I of this article, which appeared in the November issue of *COMPUTER LANGUAGE*, described their basic function but was by no means a tutorial on their use.

argc, argv, and env

First, a brief description of *argc*, *argv*, and *env* in order, since knowledge of these UNIX operating system features is fundamental to an understanding of the

PSCCS interface.

In the C language under UNIX System V, every procedure *main* has three arguments available to it when it begins executing. If we want to use these arguments, we start the definition of our procedure *main* as follows:

```
main(argc, argv, env)
int argc;
char *argv[];
char *env[]; {
```

argc is a count of how many arguments were given on the command line that invoked the process containing *main*. For example, if the invocation line is "pget -e cat", then *argc* would be 3. *argv* is a pointer to an array of strings that correspond to these arguments. In the example, *argv[0]* would be "pget"; *argv[1]* would be "-e" and *argv[2]* would be "cat". Figure 1 illustrates this arrangement. In this way a program can know by what name it

was invoked and with what arguments.

This fact is important to PSCCS since it is necessary for a program like *pget* to modify its command line arguments and pass them on to the utility *get*. The third string, *env*, is analogous to *argv* but instead of command-line arguments, it contains "environmental variables" and their values. These are shell variables known and exported by the shell from which the program was invoked. For example, if we type (in Bourne shell notation):

```
DBFILE=util
DBDIR=/port/db
export DBFILE DBDIR
```

and then

```
pget -e cat
```

env[0] would be the string "DBFILE=util" and *env[1]* would be the string

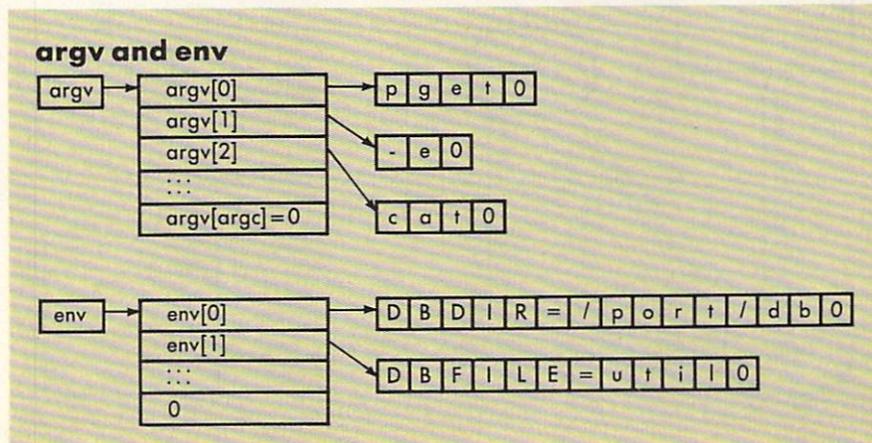


Figure 1.

"DBDIR=/port/db". (There is no count analogous to *argc* supplied for environmental variables.) This setup is also depicted in Figure 1. In this way a program can also know something about the environment from which it was executed.

PSCCS also uses this capability. For example, a user might have

```
DBDIR=/port/db
export DBDIR
```

in his or her .profile file (analogous to the Berkeley .login). Then a *pget* executed later in the session would know that the project data base file is /port/db. A C library routine called *getenv()* is supplied in System V to search the *env* array for a particular environmental variable.

Data flow in the PSCCS system

Figure 2 shows the data flow resulting from the user typing the command *pget -e cat*. Recall that the intention of this command is to find some SCCS file in the project administration area with the string

"cat" as part of its name and return the top-level version of that file to the user's current directory.

In this example we are assuming the file we want to find is /port/src/cmd/s.cat.c, and there are no other path names of files under PSCCS control with "cat" as a substring. (We will see later exactly what happens if more than one file has the given substring.)

The arrows in Figure 2 represent data flow, the oval boxes represent processes, the rectangular boxes represent files, and the numbers represent the chronological sequence of the data transfers. The files shown on the left side of Figure 2 constitute a simple data base of PSCCS path names. We will call these files the data base files. Each file is simply a list of path names, one per line, in arbitrary order, of SCCS files that the programmer may want to *get*.

The data base files are all in the same directory, in this case the directory /port/db. Although not essential, it is a good idea to group the path names into files on some basis, such as one file for

utility path names, another for path names of kernel-related files, one for game source files, etc. The reason for grouping the path names into multiple data base files will be explained later.

The two files at the right of Figure 2 are the SCCS file /port/src/cmd/s.cat.c containing the record of source code changes to *cat.c* and the source file *cat.c* obtained from this SCCS file.

Here is a step-by-step description of the data movements that occur when the user types *pget -e cat*:

- Typing the command *pget -e cat* at the terminal causes the shell to invoke the executable PSCCS utility *pget* and pass it the strings "-e" and "cat" as command line arguments.

Something else also happens during this step, which is not shown in the diagram; as described before, *pget* checks its *env* strings for the value of the environmental variable *DBDIR*. This is the name of the PSCCS data base directory containing one or more files that together contain a simple list of path names of all files under PSCCS control. In this case *DBDIR* is equal to /port/db. There is also an optional *DBFILE* environmental variable, but we will ignore it for this first pass description.

- The program *pget* uses the UNIX library routine *popen* to invoke the utility *grep* to search for the string "cat" somewhere within the ASCII files in the directory /port/db and to return to *pget* via a pipe any lines in those files that contain that string. This is equivalent to invoking the command *grep cat /port/db/**.

- As *grep* executes the search, the line "/port/src/cmd/s.cat.c" is found in the PSCCS path name file /port/db/util.

- This path name, /port/src/cmd/s.cat.c, is returned to *pget* via a pipe.

- Now that *pget* knows the full path name of the file it is looking for, it executes—via the system call *exec()*—the command *get -e /port/src/cmd/s.cat.c*.

- The SCCS utility *get* processes the file /port/src/cmd/s.cat.c to extract its top-level version for editing.

- The utility *get* writes this top-level version to the current directory as file *cat.c*. This should be what the user wants.

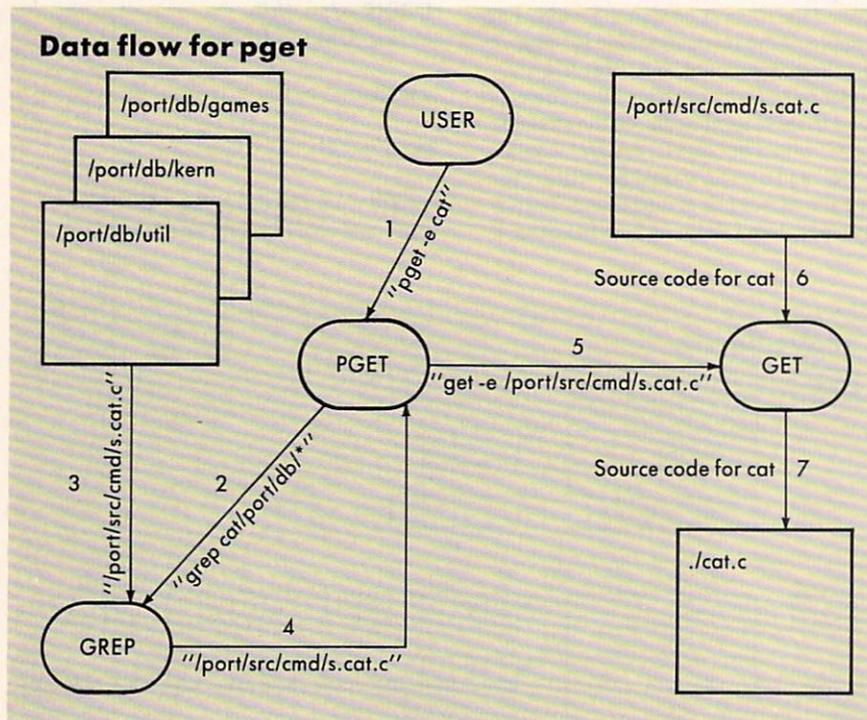


Figure 2.

The PSCCS code

Procedure *main* of the compilation unit *pget.c* is shown in Listing 1. As you can see, it is fairly trivial. Most of the real work is done in procedure *expandargs* which modify the *argv* array by expanding each substring of an SCCS file into a complete appropriate path name. For example, if the *argv* array is originally

```
pget
-e
cat
venture
```

then after calling *expandargs* it will become

```
pget
-e
/port/src/cmd/s.cat.c
/port/src/games/s.adventure.c
```

Exactly how *expandargs* does this will be discussed later. In the next statement the *0th* argument is changed to *get*. Finally */usr/bin/get* is executed via an *execv()* system call with the new set of arguments. This process is exactly equivalent to executing the command

```
get -e /port/src/cmd/s.cat.c /port/
src/games/s.adventure.c
```

by typing it in at the terminal. Since *execv* transforms the old process into the new one, it should not be possible to get to the next instruction of the old process, hence the error message on the next line.

pcdc, *pdelta*, *pprs*, *prmdel* all work in the same way as *pget*. It is even possible (and would save space) to make them all exactly identical by setting up a single program that acts differently depending upon the name it is called. This is left as an exercise for the reader. But be careful. In some cases it may not be a good idea to combine some of the utilities. For example, it may be permissible to *pget* version 1.4 of a file but not permissible to remove that version via *prmdel*. This example illustrates that the PSCCS utilities can be used to inhibit (or automatically add) certain command line options to the underlying SCCS tools. This capability is important to the *padmin* utility as we will explain below.

Listing 2 gives a pseudo-C code

description of the operation of procedure *expandargs()* along with the procedure *expand1()*, which it uses. The following is a narrative description of the operation of that code.

The first thing that procedure *expandargs* does is to check the environment string array *env* to see if a value has been assigned to the variable *DBDIR*. The data base files reside in this file. If *DBDIR* is missing, then an error exit is taken, but that is not shown in the code. Each argument in the *argv* array except for *argv[0]* and the option arguments (those whose first character is a minus sign) is then processed in turn.

Here the optional *DBFILE* environment variable comes into play. If the user has set, say “*DBFILE=util*”, this says that the user is primarily interested in retrieving utility code rather than kernel or compiler code and that therefore the data base file *util* in the *DBDIR* directory should be searched first. If a suitable path name is not found there, then the rest of the *DBDIR* directory is searched. The routine *expand1* does these searches.

expand1 invokes the *grep* utility to search the specified file or files for occurrences of the string given by *arg*. If exactly one path name is found which contains *arg* as a substring, *arg* is expanded into that path name. If more than one such path name exists, the user is asked which one is wanted.

All of these operations are fairly straightforward in C. One warning, however—don't try to expand *argv[i]* by copying the new path name character-by-character into the place pointed at by

argv[i]. The original *argv* is on the stack and there is not enough room there. Just put the path name string into some static array and redirect *argv[i]* to point there.

This completes the discussion of the code for all of the PSCCS utilities except *padmin*. The *padmin* command is significantly different from the others since it needs to update the data base files. When invoked, *padmin* must process its arguments. Like *pget*, it must call *expandargs* on all arguments except command options (arguments beginning with a “-”). In addition, *padmin* must review these command options to guarantee that the SCCS administrative parameters are set correctly.

For example, an SCCS file contains information that describes which users are permitted to access that file. The SCCS *admin* command permits the user to change this information. If unrestricted, the *padmin* command could let a user add another (unauthorized) name to the permission list. Since the integrity of the project SCCS files in part rests on restricted access to them, then that access must be protected. Hence, *padmin* must guarantee that certain options to *admin* are prohibited. Currently we prohibit the following *admin* options:

```
-fb      -dj      -r
-fj      -df      -a
-ff      -di      -z
-fi      -dq      -h
-fq      -e
```

Since the SCCS *admin* command can be

```
main( argc,
      argv, env )
int   argc;
char  **argv;
{
    expandargs( argc, argv );
    argv[0] = "get";
    execv( "/usr/bin/get", argv );
    fprintf( stderr, "exec failure" );
}
```

Listing 1.

used to create new SCCS files (the "-i" and "-n" options) and change the administrative options of existing files, then *padmin* must handle both cases. When simply changing the parameters for existing SCCS files, *padmin* works like *pget*. However, when creating new files, *padmin* has much more work to do.

The first step that *padmin* must take when creating a new SCCS file is to sup-

ply the *admin* command with the default creation options. We currently supply the following default options:

- **-rR** where R is a revision level. This option sets the current revision of the new file to the same level as other files under PSCCS control. Controlling the revision level is important since reproducing earlier versions of a product will be based on these levels. If, for example, a new file is

created at revision level 1.1, it will confuse anyone attempting to rebuild revision 1 of the system.

- **-fi**. This option requires that every PSCCS file have at least one ID keyword sequence. The "%W%Q%" sequence discussed in our first article is an example of an ID keyword sequence. With this option we guarantee that every version of an SCCS file will be generated with ver-

```
expandargs ( argc ,argv ) {
  get DBDIR from environment
  for i = 1 to argc-1 {
    if first char of arg[i] is not a minus sign {
      if there is a DBFILE environment variable {
        expandl( arg[i], DBDIR/DBFILE)
      }
      if expandl failed to expand arg {
        expandl( arg[i], DBDIR)
      }
    }
  }
}

expandl ( arg ,filenam ) {
  if filenam is a directory {
    grep for arg in filenam/*
  }else{ /* filenam must be a file*/
    grep for arg in file filenam
  }
  if exactly one pathname is found by grep {
    set current arg string to that pathname
    return success
  }else if more than one pathname is found {
    for each pathname found {
      display pathname at terminal followed by question mark
      accept answer from user (yes or no)
      if answer is yes {
        set current arg string to current pathname
        return success
      }
    }
  }
  return failure
}
```

Listing 2.

sion identification information. Knowledgeable programmers could devise a way to circumvent this guarantee, but we can't think of any reason for doing this.

■ **-fb**. This option permits creation of branches on the PSCCS file. It is included in the default options so that a later *padmin* call will not be needed.

■ **-aG** where G is a UNIX group number. This option sets up the SCCS file so that anyone in the users group can access the PSCCS file.

Once these default options are set, *padmin* will create the new SCCS file by calling *admin*. However, unlike *pget*, *padmin* must wait around until *admin* completes. (Remember, *pget* uses the *exec* system call to overlay itself with *get*.) If the *admin* command succeeds, then *padmin* must add the name of the newly created SCCS file to the project file name data base. In our current implementation, *padmin* simply appends the name to the appropriate data base file(s) in DBDIR.

To execute *admin*, *padmin* must call the *fork()* system call to create a new process. The new process (the child process) will execute the *admin* command by overlaying itself via an *exec()* system call. The original process (the parent process) will wait for the child to complete by making a *wait()* system call. This system call returns the child process's return status that will indicate whether the command succeeded or not.

As you may guess, the *padmin* command is the most complicated of the PSCCS commands. In it resides the real integrity of the files under PSCCS control. It controls the creation and administrative updating of all PSCCS files. The discussion on *padmin* has been somewhat simplified for this article. For a more in-depth discussion, a firm understanding of *admin* and the UNIX operating system are needed.

The setuid feature

Under the UNIX operating system, each file (including directories) has three levels of permissions associated with it. We are concerned with two here: owner permissions and other permissions. The owner permissions specify those operations the

owner of the file may perform on it. The other permissions specify which operations other users may perform on the file.

Several kinds of operations on the file may be permitted or denied at several levels. We are interested in permission to write to the file. Note that even if you do not have permission to write to a file, it still may be possible to delete it and then to replace it with something else. To do this, it is only necessary to have write permission to the directory containing the file. The ordinary SCCS uses this trick.

The other security feature of the UNIX operating system we need be concerned with here is that every executing process has both a real user ID and an effective user ID that are set equal to some (not necessarily the same) valid login ID of the system. The real user ID of a process is always the user ID of whomever executed the program. Usually the effective user ID is the same. However, if an executable file has its setuid bit on, then, whenever it is run, the effective user ID of the process is that of the owner of the file, not the one who executes it.

The PSCCS utilities are all setuid utilities owned by some administrator, such as "port". (Of course port must be a valid login name. In our system, port was the login name for the project administrator for the M68000 port of UNIX System V.) The PSCCS files and the directories that contain them deny write permission to everyone except port. Therefore the only way a user can modify the PSCCS files is through the PSCCS utilities.

This protection greatly reduces the likelihood of the files being inadvertently destroyed or corrupted. We have seen too many cases where an engineer has accidentally removed vital files. Under UNIX the only effective way to restore these files is to retrieve them from backups, but then you will lose any changes between the time that backups were made and the time the file was lost.

Enhancements and variations

As we have seen, the PSCCS tools have been built out of standard UNIX System V utilities and a little glue. The glue consists of approximately 600 lines of C code in a few SCCS preprocessors. This approach, while being exceptionally effective, is

very primitive. In principle, the underlying data base is really just a collection of regular UNIX files. The first obvious enhancement of these tools would be the use of a more powerful data base management scheme.

We hesitate to mention a full blown data base management system since its capabilities far exceed the requirements of the PSCCS utilities. (On the other hand, if you are currently running on one of the UNIX based DBMSs, you can and should take advantage of it.) Some simple extensions to the current approach would probably be more appropriate.

For example, one simple approach would be to maintain separate data base files where inclusion of a particular SCCS file into a particular data base file would be based on the name of the SCCS file (the SCCS files beginning with "a" in data base file "dba", files beginning with "b" in "dbb", etc.). This very simple method will significantly improve PSCCS throughput by permitting the search to begin by indexing to the correct data base file.

Another area of overall improvement in the PSCCS package is the increase in the access speed to files based on external information. For example, in the UNIX System V shell, the environment variable *CDPATH* is used to determine which directory will be used when changing directories. When using *CDPATH*, directories in non-local file systems can be accessed as if they were in the current directory. That is, if *CDPATH* is defined as

```
CDPATH = :$HOME
```

when a change directory command is issued, if the target directory is in the current directory or in the home directory, the change directory will succeed. In this same manner we can define a *SCCSPATH* that will tell the PSCCS commands to search some set of directories first before looking into the data base. For example, if *SCCSPATH* is defined as

```
SCCSPATH = dir1:dir2:dir3
```

if we execute `pget s.cat.c` while `s.cat.c` is in one of `dir1`, `dir2`, or `dir3`, then `pget` can retrieve the file without ever reading the file data base.

The `expandargs` procedure can be modified to look in the directories specified by `SCCSPATH` to see if the requested SCCS file is present. Note that this improved accessing efficiency in no way bypasses the built-in protection of the PSCCS tools.

Another interesting extension to the PSCCS tools would be to "associate" groups of PSCCS tools. Once a file in an association group is updated, the PSCCS command `pdelta` could send out electronic messages to all pertinent users that the file has been changed. This message could also contain reminders to review all associated files as well.

This feature could be best used as a way to insure that the various representations of a program are all kept in synchronization. For example, the functional specification, design specification, pseudocode design, and end product implementation can be placed into an association. If the implementation is changed, then the project administrator can be notified. If the associated specifications

and design documents are not also changed within some predetermined time frame (say one or two days), the programmer can be reminded to do so. Having this capability will greatly help project librarians keep design documents and implementations synchronized.

Other enhancements or improvements are possible. However, keeping the PSCCS tools relatively simple will take careful planning and implementation. The greatest danger of these utilities is that programmers and project administrators alike may view them as a substitute for a well-designed DBMS. They aren't. What they do provide is an economical method of insuring that project SCCS files are safe.

Benefits

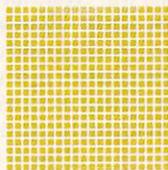
We have presented an outline of the design of a simple user interface to the UNIX SCCS. The interface has been set up to enhance the security, convenience, and control features of the ordinary SCCS. It greatly facilitated our execution of a UNIX System V port, a project involving 15 people and 6,000 files.

With the information we have presented it should be possible for a knowledgeable C programmer familiar with UNIX to set up such a system in a week or so, incorpo-

rating additional features as appropriate for a particular project management environment. ■

Luke C. Dion holds a B.S. in mathematics and computer science from the Univ. of California at Berkeley and is part way to a M.S. in Computer Science from Stanford Univ. Last February Dion left Motorola, where he was project manager and responsible for Motorola's port of UNIX system V, and founded Palomino Computer Systems Inc., specializing in UNIX operating system consulting and porting.

Alan Filipski holds a Ph.D. in computer science from Michigan State Univ. He has taught at Central Michigan Univ. and Arizona State Univ. and is currently a principal staff engineer at Motorola Microsystems in Tempe, Ariz., working on the UNIX System V operating system.



Use ALL the Power of Your MS-DOS, IBM PC-DOS, or CP/M-80 System with UNIX-Style Carousel Tools



```
ch "CP/M" "MS-DOS" <doc>newdoc
diff newdoc doc | more
ed newdoc
kwic newdoc | sortmrg | uniq | unrot >index
make -f makdoc ndx
```

Carousel Tools and Carousel ToolKits are trademarks of Carousel MicroTools, Inc. CP/M is a trademark of Digital Research; IBM is a trademark of International Business Machines; MS is a trademark of Microsoft; UNIX is a trademark of Bell Laboratories.

CAROUSEL TOOLS are a proven set of over 50 programs designed to be used with pipes, redirected I/O and scripts. In the style of UNIX each Tool does one thing well, and the Tools can be used together to do more complex tasks.

YOU ACCOMPLISH MORE using Carousel Tools: better programming and documentation support, simpler data and file housekeeping, more general file handling.

TOOLS FOR PC/MS-DOS 2.x AND CP/M-80 are available now. The DOS ToolKit is \$149. The CP/M ToolKit is \$249 and includes a *shell* to provide pipes, redirected I/O, and scripts. Source code is available for \$100 more.

ORDER YOUR TOOLKIT TODAY.



CALL OR WRITE:

 CAROUSEL MICROTOOLS, INC.

609 Kearney Street, El Cerrito, CA 94530 (415) 528-1300

CIRCLE 8 ON READER SERVICE CARD

What Day is It, Exactly?

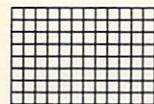
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30						

S	M	T	W	T	F	S
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31					

	1	2	3	4	5
6	7	8	9	10	11
13	14	15	16	17	18
20	21	22	23	24	25
27	28	29	30	31	

Programming a calendar using the military format and Julianized dates

By Joe Celko



Almost any system needs to use dates. But the trouble is that our calendar is not a regular, orderly measurement of time. A week is seven days, but seven is a prime number. The months do not all have the same number of days—they average 30.4375 days or 4.33 weeks. A year is not really 365 days long, it's 365 and about one-quarter days. We use years marked B.C. and A.D., but this scheme has no zero year.

These irregularities are the reason that the only unit of time defined in the metric system is the second. All other time units are considered common units, but they have no formal definition. Computer people are used to units of time smaller than a second—millisecond, microsecond, nanosecond, etc. Technically, we can use kiloseconds and gigaseconds for longer units. But nobody does, of course, because weeks and days are so much more natural.

Converting dates

Algorithms have been developed to convert dates into numbers for ease of use by the machine. This procedure is called Julianizing a date. Most of the time the numbers fall between 1 and 366, but it is just as easy to assign numbers over a range of many years as it is over one year. If you look on the bottom of many desk calendar pads, you will see the number of days past and remaining in the current year. This is one way of doing the job in the 1 to 366 range.

Computer people often confuse a Julianized date with the Julian date. The Julian date is a number well over 2 billion used by astronomers for calculations. The

algorithm for calculating it is available but useless to most programmers because of the large numbers involved.

The format for writing dates on a computer is defined in the ANSI X3.30-1971 standard. This standard allows you to use the year or the year-in-century. The year is the full four digits (such as 1984) and the year-in-century is the last two digits (84). All fields are numerics—you use leading zeros and not blanks. Months are numbered from 01 for January to 12 for December. The days are numbered in the usual fashion starting with 1 and going to 28, 29, 30 or 31 depending on the month in which they fall. The order of the fields is year, month, and day and no separators are used, which lets you sort dates as if they were one long number.

While it is not required by the standard, you should use the full year and not the year-in-century. There is no longer a great need to save computer storage space by cutting off characters, and the year 2000 is not that far away. If you use the year-in-century, you set in motion systems that will collapse in the future. Imagine the fun of having a program that computes interest based on negative time intervals.

Leap year calculations are generally a little messier than people think. Most people know that every year divisible by four

is a leap year. But most forget that there is also a 400-year cycle. Since the year 2000 is coming up shortly, it is a good idea to consider it. The four-year cycle exists because the number of days in a year is 365 and almost one-quarter. Leap year takes care of the one-quarter day, and the 400-year rule takes care of the "almost." It is a very fine adjustment.

Listing 1 presents a simple algorithm in an ALGOL-like pseudocode for testing a leap year. It is not very confusing, and the cost of the extra Boolean expression to compute the 400-year cycle is tiny. Although few people work with time that goes over centuries, I like to get the accuracy when it is so cheap.

Error checking is always a problem, and you should always try to catch errors as soon as possible in a system. Since dates are messy things, they will need to pass through an edit procedure before they get to the data base. It is worth the effort to have reasonableness edits at places in the program.

The terms reasonable lower limit and reasonable upper limit have to be user defined. If the system does not allow front dating, then today's date is a good value for the reasonable upper limit. If the sys-

```

BOOLEAN PROCEDURE Leapyear (INTEGER year);
/* year is four digits, not year-in-century */
Leapyear:=
((year MOD 4 = 0) AND (year MOD 400 <> 0))
OR (year MOD 400 = 0);

```

Listing 1.

January						
S	M	T	W	T	F	S
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31					

tem did not exist before some date, then it might be a value for reasonable lower limit. A more complex situation (Listing 2) might involve the relationships among the dates for these bounds (you cannot fire an employee before you hire one and so forth).

Date formats

We still need to get dates into the data base, which means that a human being must key them in. But we have yet another problem. Nobody agreed on a way to

write a date. The most common formats you will see are with slashes and numbers, dashes and numbers, full month names, and three-letter month codes.

The following list is of possible formats, compressing out blanks. The "mm" is a numeric month, and "mmm" is a three-letter code. The "dd" is a day in the month, and "ddd" is the ordinal number of the day in the year (001 to 365 or 366).

mm/dd/yy—U.S. format

dd/mm/yy—U.K. format

mm-dd-yy—U.S. computer format

dd-mm-yy—U.K. computer format

mmmdd,yyyy—Correspondence format
 yyymmmdd—Military format
 yyymmdd—ANSI format
 yymmdd—Alternate ANSI format
 yyddd—Ordinal ANSI format

The trouble with the all-numeric dates is that U.S. and U.K. dates are easily confused. The current military date uses year, three-letter month, and day—in that order—for ease of date processing. It replaces the older military format, "ddmmmyyyy".

```

BOOLEAN PROCEDURE Validate (INTEGER year, INTEGER month, INTEGER day)

BEGIN EXTERNAL BOOLEAN PROCEDURE LeapYear;
INTEGER MinSystemYear INIT /* reasonable lower limit */,
      MinSystemYear INIT /* reasonable upper limit */
INTEGER ARRAY MonthSize[1:12]
  INIT (31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31);
IF LeapYear(year)
THEN MonthSize[2] := 29
ELSE NULL;
ValidDate:= ((month >= 1) AND (month <= 12))
  AND ((day >= 1) AND (day <= MonthSize[month]))
  AND ((year >= MinSystemYear) AND (year <= MinSystemYear));
END of ValidDate;

```

Listing 2.

```

INTEGER PROCEDURE Julian (INTEGER year, INTEGER month, INTEGER day);
/* This is Julianized day within year, not an astronomer's Julian day

BEGIN
EXTERNAL BOOLEAN PROCEDURE ValidDate, LeapYear;
IF ValidDate (year, month, day)
THEN BEGIN
  INTEGER ARRAY MonthTotal[1:12]
    INIT(90, 31, 59, 90, 120, 151, 181, 212, 243, 273, 304, 334);
  IF (LeapYear(year) AND (month > 1))
  THEN Julian:= 1
  ELSE Julian:= 0;
  Julian:= Julian + MonthTotal[month] + day;
ELSE Fault ("This is a bad Date");
END of Julian;

```

Listing 3.

December

S	M	T	W	T	F	S
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

I strongly recommend the current military format. It is much easier for a human being to read correctly. It takes no trouble at all to convert between ANSI and this display format. All you need is a little string handling and a table of the three-letter codes.

It is also possible to write an input procedure that will accept a wide range of date formats and convert them into "yyyymmdd". This will allow the user to type in the date in almost any format. But the procedure should immediately overwrite the input with the converted date so that the user can see how the machine interpreted what was typed in. Remember, there is no way to tell if the U.S. "mm/dd/yy" or the U.K. "dd/mm/yy" format was meant, so you have to make an assumption based on your geography.

Examples

Now we are ready to Julianize a date with the procedure presented in Listing 3.

This approach can be extended over a wider range by adding a year total table, which would be indexed by the year minus a constant. The tables are fairly small and represent a good trade-off with approaches that would use floating point

calculations. This algorithm allows you to generate ANSI ordinal format dates by attaching the year-in-century to the high order digits of the result.

But algorithms do not have to be unique. Just take a look in any textbook at all the ways to sort an array. Another algorithm that can perform this same function is shown in Listing 4. It is based on smoothing out the distribution of days in the year (365.25 days per year) and then using a set of *IF-THEN-ELSE* statements to put the irregularities back into the calculation to adjust the date.

For most people, there is no way to look at a day and tell what day of the week it fell upon. The algorithm in Listing 5 computes the day of the week from a date given in the format that we have been using for the Julianization algorithms.

In Listing 6, pseudocode for computing the number of days between two dates is presented. It depends on an encoding algorithm that returns a pseudo-Julianized date. The pseudo-Julianized date is a real number that represents the position of the day within the year if all months and days were spread uniformly over a year. It is not a completely accurate Julian calculation because of these assumptions, but the differences are accurate enough for computing interest to a few places and other financial applications.

Commercial users often write programs under the assumption that the year has 360 days and all months have 30 days. This algorithm is an improvement over the common practice.

If you ever get a chance to vote for calendar reform in your lifetime, please do so. Several schemes have been proposed, all superior to the present system. I favor the Edwards calendar, which has four three-month quarters. The months are 30, 30 and 31 days long in each quarter. This scheme splits New Year's Day away from any month and uses a zero to represent it. Leap year day appears every four years in the middle of the year and is also shown with a zero day at the start of the third quarter.

It is left as an exercise for the reader to design programs for the Edwards calendar. But they are very easy. ■

Joe Celko is a contractor and consultant currently residing in Los Angeles, Calif. He was recently employed as a computer research scientist at Georgia Tech in Atlanta, Georgia. Celko is a regular columnist for Information Systems News and Software News.

```

INTEGER PROCEDURE Jul2 (INTEGER year, INTEGER month, INTEGER day);
/* This is Julianized day within year, not an astronomer's Julian day *

BEGIN EXTERNAL BOOLEAN PROCEDURE ValidDate,
LeapYear;
If ValidDate (year, month, day)
THEN BEGIN
    Jul2:= Truncate ((30.475 * (month-1)) + day);
    IF (month > 2) THEN Jul2:= Jul2 + 2 ELSE NULL;
    IF (month > 5) THEN Jul2:= Jul2 + 1 ELSE NULL;
    IF (month > 6) THEN Jul2:= Jul2 - 1 ELSE NULL;
    IF (month > 7) THEN Jul2:= Jul2 - 1 ELSE NULL;
    IF (LeapYear(year) AND (month > 1))
    THEN Jul2:= Jul2 + 1 ELSE NULL;
    END
ELSE Fault ("This is a bad Date");
END of Jul2;

```

Listing 4.

```

STRING PROCEDURE DayOfWeek (INTEGER year, INTEGER month, INTEGER day);
/* returns the three-letter day of week */

BEGIN EXTERNAL BOOLEAN PROCEDURE ValidDate, LeapYear;
STRING ARRAY DayName [0:6]
  INIT ("SUN", "MON", "TUE", "WED", "THU", "FRI", "SAT");
INTEGER ARRAY DayTable[1:12]
  INIT (0, 3, 3, 6, 1, 4, 6, 2, 5, 0, 3, 5);
INTEGER ARRAY CenturyTable[1:12]
  INIT (1,2,0,6,4);
INTEGER Century, YearInCentury, Result;
IF ValidDate (year, month, day)
THEN BEGIN
  IF (LeapYear(year)
  THEN BEGIN
    DayTable[1] := 6;
    DayTable[2] :=2;
    END
  ELSE NULL;
  Century:= Year DIV 100;
  YearInCentury:= year - (Century * 100);
  Result:= CenturyTable[Century] +
    YearInCentury +
    (YearInCentury DIV 4) +
    DayTable[month] +
    Day;
  DayOfWeek:= DayName [Result MOD 7];
  END
ELSE Fault ("This is a bad Date");
END of DayOfWeek;

```

Listing 5.

```

INTEGER PROCEDURE DateDiff (INTEGER year1, INTEGER month1, INTEGER day
  INTEGER year2, INTEGER month2, INTEGER day2);
BEGIN
REAL PROCEDURE Pseudo-jul (INTEGER year, INTEGER month, INTEGER day);
/* this will give a Pseudo-Julianized date */
BEGIN EXTERNAL BOOLEAN PROCEDURE ValidDate;
IF ValidDate (year, month, day)
THEN Pseudo-jul :=
  ((365.25 * year) + (30.475 * month) + day)
ELSE Fault ("This is a bad Date");
END of Pseudo-julian;

EF ValidDate (year1, month1, day1)
THEN IF ValidDate (year2, month2, day2)
  THEN DateDiff:=
    Truncate (Pseudo-jul (year2, month2, day2)
      - Pseudo-jul (year1, month1, day1)
      + 0.05)
  ELSE fault ("Second date is bad");
ELSE fault ("First date is bad");
END DateDiff;

```

Listing 6.

Customize Your High-Level Language

A COBOL-assembly example

By Charles Ballinger

 Although this article deals with the specifics of interfacing Microsoft's COBOL-80 language with assembler subroutines, the concepts and requirements are applicable to a wide range of other high-level languages.

The programs discussed were written and tested using Microsoft's COBOL-80 compiler version 4.60 and 4.65. The assembler used was Microsoft's M80, which is included with the COBOL compiler, but any assembler capable of producing an .REL file can be used (i.e., RMAC, etc.).

Why would you want a high-level language to interface to an assembler module? I realized I wanted such a high-level language interface when I was writing an application program in COBOL and needed to control special printer functions such as spacing and font size.

Although COBOL is an excellent business language, it has several shortcomings. One of them is its inability to easily pass embedded control codes to the printer.

At this point I set out to find the information necessary to write an assembler subroutine that my COBOL program could call to handle these special printer functions. Although the assembler subroutine deals with an Okidata 83A printer, it can be easily changed to support whatever printer you are using.

The Microsoft COBOL-80 manual is very skimpy in the areas of sample programs. Only a mention is made of the

method that must be employed when attempting to call an assembler subroutine.

Most of my discoveries involved nothing more than trial-and-error attempts to discover what the magical linkage should be. This article can be a road map that takes you on a much shorter journey than the one I took. By the end of the article you should be able to see just how easy the interface is and how easily the assembler program could be modified to support a similar function you have always wanted.

You will undoubtedly see areas in the assembler source that can be shortened or eliminated entirely. The assembler program is heavily commented so that the COBOL programmer who knows a little assembler can readily modify the code to suit his or her needs. I will not attempt to teach you either COBOL or 8080 code in this article but instead will show you the things I uncovered that do not appear in any Microsoft sample program and are only hinted at in its documentation.

Microsoft's mention

To start off, let's review just what Microsoft's reference manual has to say about calling an assembler program. Everything the manual contains on this subject can be found in Appendix B of the COBOL-80 reference manual.

This appendix briefly describes the interface calling mechanism used when COBOL calls an assembler subroutine. The COBOL run-time system transfers control to the subroutine by means of the machine language *CALL* instruction. The subroutine must return control to COBOL by issuing a machine language *RET* instruction.

Parameters are passed from the high-level language down to assembler by reference—by passing the address of the data and not the data itself. The method of passing these parameters depends on the number of parameters you are attempting to pass. If there are from one to three parameters to be passed they are passed in the following manner:

Parameter 1 in register pair HL
Parameter 2 in register pair DE
Parameter 3 in register pair BC

If you are passing more than three parameters, parameters 1 and 2 are still passed in HL and DE, but register pair BC now points to a contiguous data block of memory that holds the list of parameter addresses. Remember, if you pass more than three parameters, this contiguous block of memory contains the addresses of the additional parameters in byte-reversed order.

Also note that neither the compiler nor the run-time system checks to ensure that the correct number of parameters or their sequence has been passed. You are responsible for the integrity of the parameters as well as matching up the sequence in both the calling and the called program. If you get weird results, this should be the first thing you check.

Since the stack space used by COBOL is contained within the program boundaries, the assembler program must not

```

IDENTIFICATION DIVISION.
PROGRAM-ID. TEST.
AUTHOR. CHUCK BALLINGER.
DATE-WRITTEN. 04/10/84.

```

```

*
* THIS PROGRAM IS A DEMO PROGRAM TO SHOW YOU HOW THE
* CALL TO "PRSET" PROGRAM WORKS. THE USE OF ASSEMBLER
* SUBROUTINES ARE ONLY LIMITED BY YOUR IMAGINATION.
*

```

```

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER.
OBJECT-COMPUTER.
INPUT-OUTPUT SECTION.
FILE-CONTROL.

```

```

    SELECT PRINTL ASSIGN TO PRINTER.

```

```

DATA DIVISION.
FILE SECTION.

```

```

FD PRINTL
   LABEL RECORDS ARE OMITTED
   RECORD CONTAINS 132 CHARACTERS
   DATA RECORD IS PRINTLN.

```

```

01 PRINTLN.
   02 FILLER                      PIC X(132).

WORKING-STORAGE SECTION.
77 FUNCTION-CODE                  PIC X VALUE SPACE.
77 RETURN-CODE                    PIC X VALUE SPACE.
01 TESTLN-1.
   02 PART1                       PIC X(10).
   02 PART2                       PIC X(10).
   02 PART3                       PIC X(10).
   02 PART4                       PIC X(10).
   02 PART5                       PIC X(10).
   02 FILLER                      PIC X(82) VALUE SPACES.

```

```

PROCEDURE DIVISION.

```

```

START-OF-PROGRAM.

```

```

    OPEN OUTPUT PRINTL.

```

```

*
* SET PRINTER TO 8/LINES INCH AND PRINT DEMO
*
    MOVE "8" TO FUNCTION-CODE.
    MOVE " " TO RETURN-CODE.
    CALL 'PRSET' USING FUNCTION-CODE, RETURN-CODE.
    IF RETURN-CODE = " " NEXT SENTENCE ELSE
        GO TO ERROR-OUT.
    MOVE SPACES TO PRINTLN.
    MOVE "THESE LINES PRINTED AT 8 LINES PER INCH" TO PRINTLN.
    WRITE PRINTLN AFTER ADVANCING 1 LINES.
    WRITE PRINTLN AFTER ADVANCING 1 LINES.
    WRITE PRINTLN AFTER ADVANCING 1 LINES.
    WRITE PRINTLN AFTER ADVANCING 1 LINES.
*
* SET PRINTER BACK TO 6 LINES/INCH
*
    MOVE "6" TO FUNCTION-CODE.

```

overflow or underflow the stack. To be sure this doesn't happen, the assembler program, upon entry, should save the COBOL stack and then set up its own stack pointer.

The preceding information brings to a close what the COBOL-80 manual says about calling an assembler subroutine. You probably noticed that no samples or clues are given as to just what registers are affected.

A working version

At this point frustration set in. So I called the Microsoft technical support line only to be told, "Nobody here has ever tried that." About all the Microsoft people were sure of was that you had better save every register upon entry into your assembler program because they didn't have any information about what, if any, registers were affected. With this lack of information I then proceeded through the trial-and-error sessions necessary to get a working version of the assembler subroutine.

As you can see in Listing 1, the COBOL code necessary to *CALL* an assembler subroutine is very straightforward. The calling sequence is:

```

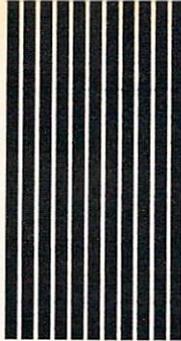
CALL 'PRSET' USING FUNCTION-
CODE, RETURN-CODE.

```

Only two parameters are used. The first, *FUNCTION-CODE*, is passed in the HL register pair, while the second, *RETURN-CODE*, is passed using the DE register pair. The program does not make use of a third parameter, so registers BC are left in an unknown state but are saved anyway once inside the subroutine.

Up to now we have just covered the basics of the linkage required. I have written a printer setup program (PRSET), which can be used to set special printer modes. (Unfortunately it is too long to be printed here, but it can be accessed by calling the *COMPUTER LANGUAGE* Bulletin Board Service or CompuServe.)

The PRSET assembler program can be refined and reduced in size if you wish to experiment. My goal in writing the program was to make it easy for a non-8080 programmer to modify. I'm no 8080



giant, but the code works, is easy to follow, and lends itself to whatever modifications you may wish to implement.

The subroutine only has a few areas you must watch for. If you are writing your own routine, be sure that at the beginning of the program you have an *ENTRY* statement program to name the module. At link time this is required to name the module so the COBOL linker can find and link it to the main COBOL program. This program uses the standard entry points for CP/M 2.2+ for both BDOS and LIST devices. If your particular version of CP/M uses an entry of other than 05H (5 Hex), then change the equates in the PRSET program.

The first thing the program does is save the HL register pair. Then it swaps HL and DE and finally saves HL again. The first *SHLD* saves parameter 1's address. The registers are then swapped and the second *SHLD* saves parameter 2's address. The program then saves register pair BC even though they are not used.

Here's the tricky part. You must clear the HL register pair and then do a *DAD SP* instruction to load the value of the COBOL stack pointer. Now the program saves the COBOL stack pointer in a field called COBSTK and proceeds to establish its own stack area. Register pair HL is then loaded with the address of parameter 1, and the program checks to see what, if any, option was requested.

If no valid option was found, then the code falls through to the error routine. At the error routine the program gets addressability to the return code field (parm 2) and moves an X into the return code field to indicate it found an error. The registers are then restored, and the COBOL stack pointer is restored to the same value it had upon entry. A machine language *RET* instruction is then executed, which returns control back to the COBOL program.

The normal exit point in the program (0210H) does the same thing with the exception that the return code field is set to a blank prior to returning.

The individual routines within the sub-

```

MOVE " " TO RETURN-CODE.
CALL 'PRSET' USING FUNCTION-CODE, RETURN-CODE.
IF RETURN-CODE = " " NEXT SENTENCE ELSE
    GO TO ERROR-OUT.
MOVE SPACES TO PRINTLN.
MOVE "THESE LINES ARE AT 6/LINES PER INCH" TO PRINTLN.
WRITE PRINTLN AFTER ADVANCING 2 LINES.
WRITE PRINTLN AFTER ADVANCING 1 LINES.
WRITE PRINTLN AFTER ADVANCING 1 LINES.
WRITE PRINTLN AFTER ADVANCING 1 LINES.
*
* PRINT AT VARYING FONTS SIZES ON SAME LINE
*
MOVE "C" TO FUNCTION-CODE.
MOVE " " TO RETURN-CODE.
CALL 'PRSET' USING FUNCTION-CODE, RETURN-CODE.
MOVE SPACES TO PRINTLN, TESTLN-1.
MOVE "16.5 CPI" TO PART1.
WRITE PRINTLN FROM TESTLN-1 AFTER ADVANCING 3 LINES.
MOVE SPACES TO PART1.
MOVE "B" TO FUNCTION-CODE.
MOVE " " TO RETURN-CODE.
CALL 'PRSET' USING FUNCTION-CODE, RETURN-CODE.
MOVE "8.3 CPI" TO PART2.
WRITE PRINTLN FROM TESTLN-1 AFTER ADVANCING 0 LINES.
MOVE SPACES TO PART2.
MOVE "W" TO FUNCTION-CODE.
MOVE " " TO RETURN-CODE.
CALL 'PRSET' USING FUNCTION-CODE, RETURN-CODE.
MOVE "5 CPI" TO PART3.
WRITE PRINTLN FROM TESTLN-1 AFTER ADVANCING 0 LINES.
MOVE SPACES TO PART3.
MOVE "R" TO FUNCTION-CODE.
MOVE " " TO RETURN-CODE.
CALL 'PRSET' USING FUNCTION-CODE, RETURN-CODE.
MOVE "10 CPI" TO PART4.
WRITE PRINTLN FROM TESTLN-1 AFTER ADVANCING 0 LINES.
END-THE-PROGRAM.
DISPLAY (24, 1), "RETURNING TO CP/M - PLEASE STANDBY" ERASE.
CLOSE PRINTL.
STOP RUN.
*
* DISPLAY ERROR IF INVALID RETURN CODE WAS FOUND
* YOU DECIDE HOW YOU WANT TO HANDLE IT. ONCES TESTED
* YOU MAY WANT TO IGNORE THE RETURN CODE FIELD
*
ERROR-OUT.
DISPLAY (24, 1) ERASE
"YOU WERE PASSED A RETURN INDICATING AN ERROR OCCURED"
GO TO END-THE-PROGRAM.

```

Listing 1. (Continued from preceding page).

routine can be expanded or combined if you wish. The program currently accepts any number of characters to pass to the list device, so you are not limited as to how many control functions you can do within one *CALL*. The program will terminate when a byte containing zero hex (00H) is encountered. As long as you terminate each table entry with this value you can set multiple conditions in each routine.

Now that I had finally succeeded in getting COBOL and assembler to talk to each other, the program didn't seem that hard. As long as you save all your registers

upon entry and restore them before you leave, you can use this subroutine to do many things besides control your printer. Perhaps you wanted to do special screen graphics to produce a form but didn't like the idea of trying that in a high-level language. With this subroutine you can now do it easier and faster.

Figure 1 shows the link statement necessary to link the TEST program with the printer setup program (PRSET). Figure 2 illustrates the assembler instructions necessary to assemble the PRSET.MAC program into a .REL file that is ready to be linked to the main COBOL program.

As you can see from this article and listings, the task really wasn't that hard. The only major drawback was the lack of truly

technical information of any kind.

I am releasing these programs into the public domain, and it is my hope that you can make use of the information presented here. ■

Charles Ballinger is a systems analyst and has been involved in computers and programming for a little over 10 years. He has been actively involved in microcomputers since 1978.

```
LD80 TEST/N,TEST,PRSET/E
```

Subroutine name followed by /E to indicate that this is the end of the load and to return to operating system when finished with link.

The first module to be linked, this MUST be the main COBOL program or you will get link errors.

This specifies the name under which this program is to be stored. The /N indicates this is what you want the program named to the linker.

If you have more than one assembler subroutine that you wish to call from COBOL then you would list them like this:

```
LD80 TEST/N,TEST,SUB1,SUB2,PRSET/E
```

This would link the program TEST with the subroutines named SUB1, SUB2, and PRSET. You may specify the subroutines in any order and the linker will resolve the necessary addresses.

Figure 1.

```
M80 PRSET,LST:=PRSET
```

Name of source file, this file must have an extension of .MAC to be recognized by the M80 assembler.

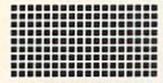
Name of printer output file or device to receive the printed output. Can specify CON: if you want to display the output.

Name of .REL module that will be generated by the M80 assembler. This is the name of the module that you would use within the COBOL link statement.

Figure 2.



PUBLIC DOMAIN SOFTWARE REVIEW



We've been having a terrific debate on the *COMPUTER LANGUAGE* section of CompuServe over the "best" programming language.

For those who haven't yet tried our CompuServe section, the conclusion to be drawn from the debate is that there is no best language. Each language has its own merits and drawbacks for any given situation, although probably two or three would do equally well for the job.

Naturally, personal biases come in to play, and those who program in only one or two languages have as their own favorites the languages they are most familiar with. Among the multi-language people there is about an even split between C, FORTH, Pascal (Modula-2), and number crunchers like FORTRAN and APL. (The latter's proponents tend to be engineers or scientists.)

It becomes very obvious to any programmer who can write code in several languages that some are general purpose by design, such as Pascal and BASIC, while others are not—APL, for example, cannot possibly be mistaken for anything else but a number cruncher at heart. So the unsolvable debate will continue to the amusement of all (and probably to the education of all, too).

What is the best language? The one *you* like the best.



With so many languages available to a programmer, it is no surprise at all that each has a growing number of support organizations. These users groups devote themselves to distribution of information or programs about a specific language, or more usually, a specific implementation of that language. Therefore, any public domain columnist must address these groups as well as the generally targeted users groups such as CP/MUG and SIG/M.

Leor Zolman's BDS C has been through many revisions over the past few years and continues to be one of the most popular C implementations available. As the

number of users of BDS C expanded, a nucleus appeared that took the form of the BDS C Users Group. The group began collating material for inclusion on its disk volumes and very quickly grew to a respectable number.

While the BDS C compiler is emphatically not public domain, the associated software from the users group is. And while the BDS C compiler is intended as the compilation system, there is no major obstacle (except some syntax and lesser known functions) to using either the public domain Small-C or another commercial C compiler on the source codes. Most of the group's disks include source code as well as compiled code.

The BDS C Users Group has not assigned a cataloging system like SIG/M and CP/MUG uses, so the contents of each disk are sometimes hard to decipher. A master catalog (which looks very much like it was done on Ward Christensen's XCAT/NCAT system) is available that lists file names with their associated disk but has no descriptions at all. (At least not in the releases I have seen—such a thing may exist now if someone has taken the time to compile it.)

Regardless, several dozen disks are available either from the users group directly (address at the end of this column along with all other mentioned organizations) or through some of the clearing houses that specialize in distribution of public domain material, such as Elliam Associates.

As BDS C Users Group disks have no accurate numbering system, an arbitrary system has been adapted by several sources. Currently 44 disks have contents varying from the ubiquitous games to software tools, utilities, functions, and other material. One of the disks even has the Small-C (public domain) compiler on it.

Notable disks have the ROFF4 text formatter, a 6800-1802 cross assembler, Ed Ream's screen editor (both BDS C and Small-C versions), video terminal libraries, 6809 software tools, and PISTOL (see last month's column). Five disks contain utility programs, and three have functions. The complete software tools are available.

Pascal/Z has also developed a users group with a similar series of disks avail-

By Tim Parker

able. Again, most of the programs are available in source code form, so any Pascal will work after suitable modification where necessary. Most of the Pascal/Z Users Group's disks have been released on SIG/M disks, so two sources exist.

One volume that always seems to raise an eyebrow is Pascal/Z vol. 14, SIG/M vol. 71, which includes among the bit manipulator, statistical package (worth the look) and randomizer, a LISP written entirely in Pascal! While it is quite a subset, the novelty alone of looking at the code (all 26K worth) is instructive to LISPer and Pascal users alike. (Now I'd like to see a Pascal written in LISP.)

Another useful volume from a machine language programmer's point of view is Pascal/Z vol. 22, SIG/M vol. 97. This has an updated COMBINE (discussed in the first Public Domain Software Review column—it combines a number of small assembler files into one larger one) and a program called CROSS, which allows indentation, annotation, and cross-referencing of Pascal code. The Pascal source code is supplied as are compiled versions.

Finally, Pascal/Z vol. 25, SIG/M vol. 133, has a metric conversion program and a balanced cattle feed ration design program. (Luckily, those aren't the best things on the disk.) An extended precision floating point routine, which has worked well in testing, is also included.

The (unfortunately) defunct JRT Systems' JRT Pascal (which pioneered the low-cost CP/M software idea) is available on Pascal/Z vol. 18, SIG/M vol. 82. This is version 2 of the compiler. JRT Pascal 3.0 is also available, but the disk numbers have not yet been announced.

Despite a lot of criticism, JRT Pascal has been the Pascal entry vehicle for thousands of programmers, and its inclusion in the public domain is bound to allow many more people to access Pascal. (JRT



Pascal 3.0 is available on CompuServe's *COMPUTER LANGUAGE* data base section along with complete documentation. An extended precision package allowing 502 digit manipulation is also there. It is not yet available through any public domain source as it is a user-supported program.)



A company called Elliam Associates has a catalog of all current CP/MUG and SIG/M disks and abbreviated descriptions available. Elliam supplies the disks in many formats and over the past few months has proven itself to be extremely

reliable and cooperative. The company is very highly recommended from personal experience and will be mentioned here quite often in the future in connection with some of its releases. Elliam's catalog costs \$7.50.

Elliam collects material of similar types on several disks. For example, Elliam has a disk of SUBMIT facilities that will collate the best (and most recent versions) of the submit programs from the disks in the library and present them on one collection. While it may be argued that getting a mass of submit programs at once is not that beneficial because so many won't be used, it must also be considered that this allows a user who wishes to use a submit program to find the one that best suits the requirements.

Elliam offers collections with several themes, including copy programs, disk utilities, directory and catalog routines, and others. Two I have found of value as a programmer deal with assemblers and disassemblers and Z80 programs (for the ol' 8-bit machines).

The assemblers and disassemblers disk contains 49 files in 420K and is available for \$22.00 in most formats (obviously on multiple disks for some). While some of the programs are of limited use to most people unless they happen to own specific systems, there is enough good material to make the collection a worthwhile addition to a CP/M-80 programmer's library.

A sampling of the included programs shows several straight assemblers. ASM_X allows cross-reference maps to be generated and recognizes Z80 mnemonics. A file includes the index register op code syntax that ASM_X handles. ASM_X isn't perfect (what program is) in that the disk buffer is a paltry 128 bytes needing many disk accesses, and it requires all labels to be in column one (instructions cannot be there!), but a label can be the same as a valid op code, if required. Six additional op codes can be handled by ASM_X to allow listings to be turned on and off, paging, conditional paging, and changes of listings from hex to octal. ASM_X doesn't handle macros, which is not too bad a problem as MACASM on the same disk does.

LINKASM is compatible with ASM from Digital Research and allows a series of files to be linked together during assembly. LINKASM seems to run faster than ASM under all conditions and allows the construction of a symbol table that can be used with SID.

Disassemblers on the disk include the previously discussed RESOURCE (see the premier issue) by Ward Christensen modified to handle Z80 mnemonics; the Intel Users Group's DISASM, which has been severely overhauled to handle Z80 mnemonics and disassemble better; and DISSAMBR, an 8080 BASIC program.

Of those mentioned, RESOURCE and its derivatives for Z80s have performed flawlessly in countless attempts to dis-

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 II, III+, //e, //c. Source code for libraries is included. **Only \$49.95**
- **ASSYST: The Assembler System** - A complete 6502 editor/assembler and linker for APPLE DOS 3.3. 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

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 minicomputer 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 & proprietary 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 compiler (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 Compiler, 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:



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

semble and understand programs. RESOURCE has a well-written document file and allows the programmer (or in this case "unprogrammer"?) enough flexibility to create the source code in whatever is best for the situation.

The disk also has a number of programs designed to make a programmer's life much easier. NOTATE and COMBINE have been previously mentioned in this column, but for those who missed it, NOTATE allows comments to be added to a source program. It displays each uncommented line with the cursor positioned at the comment location and allows the programmer to add a comment if required or to simply skip over the line. The original file is saved as a .BAK file, while the new version takes the .ASM file type.

COMBINE concatenates a number of source codes into one larger program and allows comments to be stripped at the same time. When a large program is designed in modules, this program proves itself very worthy.

Included is a structured assembler "programming language" called ML80, which seems ideal for those who want to program in machine language but can't get a grip on the mnemonics. It allows 8080 code to be written in a preprocessor style that prevents memorization of all the functions available. As an example, to increment the accumulator the line "A = A + 1" can be used instead of the standard "INR A". (That doesn't particularly excite me, but there must have been some need for the program. It does have a good macro facility.)

Finally, XREF is the source for a cross-reference generator that has been modified over previous versions to accept lower case and add extra error messages. PAGE and ELSE pseudo-ops are ignored.

Some of the other Elliam collection disks will be examined in upcoming columns, but that's enough for now. In the next column, you'll see more for Big Blue and maybe even a look at some Apple stuff. (There were a lot of Apples sold . . . they must be out there somewhere!)

Some of the programs mentioned here are available with a tub-load of languages on either the *COMPUTER LANGUAGE* Bulletin Board Service or on CompuServe. So, till next we meet, ciao!

Useful addresses: SIG/M is at P.O. Box 2085, Clifton, N.J. 07015-2085. CP/MUG is at 1651 Third Ave., New York, N.Y. 10028. PC-SIG is at 1556 Halford Ave., Suite 130, Santa Clara, Calif. 95051, (408) 730-9291. Elliam Associates is at 24000 Bessemer St., Woodland Hills, Calif. 91367, (818) 348-4278. Pascal/Z Users Group is at 7962 Center Parkway, Sacramento, Calif. 95823, and the BDS/C Users Group is at Box 287, Yates Center, Kan. 66783. 

The Preferred C Compiler

"...C86 was the only compiler we tested that ran every benchmark we tried and gave the expected results... Computer Innovations C86 was the compiler that our staff programmers used both before and six months after we conducted the tests."

J. Houston, BYTE MAGAZINE - February 1984

*FAST EXECUTION - of your programs.

*FULL & STANDARD IMPLEMENTATION OF C - includes all the features described by K & R. It works with the standard MSDOS Linker and Assembler; many programs written under UNIX can often be compiled with no changes.

*8087 IN-LINE - highly optimized code provides 8087 performance about as fast as possible.

*POWERFUL OPTIONS - include DOS2 and DOS1 support and interfaces; graphics interface capability; object code; and librarian.

*FULL LIBRARY WITH SOURCE - 6 source libraries with full source code the "large" and "small" models, software and 8087 floating point, DOS2 and DOSALL.

*FULL RANGE OF SUPPORT PRODUCTS FROM COMPUTER INNOVATIONS - including Halo Graphics, Phact File Management, Panel Screen Management, C Helper Utilities and our newest C to dBase development tool.

*HIGH RELIABILITY - time proven through thousands of users.

*DIRECT TECHNICAL SUPPORT - from 9 a.m. to 6 p.m.

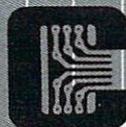
Join The Professional Programmers Who Agree C86™ Is The C Compiler Of Choice

For Further Information Or To Order Call:

800-922-0169

Technical Support: (201) 542-5920

980 Shrewsbury Avenue
Suite PW509
Tinton Falls, NJ 07724

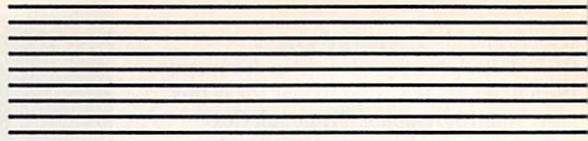


Computer Innovations, Inc.

C86™

PRICES SUBJECT TO CHANGE WITHOUT PRIOR NOTICE.
UNIX IS A TRADEMARK OF BELL LABS. C86 IS A TRADEMARK OF COMPUTER INNOVATIONS, INC. MSDOS IS A TRADEMARK OF MICROSOFT.
PCDOS IS A TRADEMARK OF INTERNATIONAL BUSINESS MACHINES.

EXOTIC LANGUAGE OF THE MONTH CLUB



OMNI: One person's language

By Steve Heller

In 1979, after I had spent two years writing a general data base program in a version of Microsoft BASIC, with ad hoc extensions in Z80 assembly language, I decided that no programmer should ever have to go through the same ordeal. OMNI is the result of that determination. OMNI's purpose is to provide the professional programmer with the tools needed to write commercial programs quickly and easily. These tools include powerful and flexible data types, including variable-length virtual memory string arrays; a uniform and simple syntax—reverse polish notation with no hierarchy of operators to remember; and a highly interactive environment, including a word-processor-type program editor and an exceptionally rapid compiler.

It has taken me the equivalent of more than two years of full-time work to create the prototype of the OMNI system. Table 1 compares in detail the features of OMNI with other currently available languages.

Since I perceive the greatest market for microcomputer programming to be in commercial data processing, OMNI's design was heavily influenced by the requirements of such programming. However, other types of programming, such as scientific programming, should also benefit from the highly interactive nature of the programming process in OMNI.

Some of the major design criteria (and their solutions) are:

- Extremely rapid compilation to make testing and debugging an interactive task. This relies heavily on three concepts: hash coding of the symbol table; segregating special compiling functions into the operators that need them, thus reducing the checking that must be done in the compiler proper; and using reverse polish notation to reduce the compiler's job essentially to looking up the indexes of the operations whose names are listed in a definition.
- Compact object code to allow more room for data in memory and to reduce virtual memory overhead for code swapping. In order to achieve this, I allow for

two different sized operators: 1 byte for primitives, which are defined in assembly language, and 2 bytes for secondaries, which are defined as a sequence of primitives and/or other secondaries. Since the most common operations are primitives, this reduces the average object code size to less than 2 bytes per operator.

- Good execution speed for real world programs to promote commercial acceptance of programs produced. This was accomplished by coding the most commonly used functions in assembler rather than attempting to increase the portability of the language itself by coding as much as possible in higher-level code as is done in Fig-Forth, for example.

- Excellent debugging facilities at a high level so that machine language debugging is not necessary.

- Excellent run-time error checking. For example, all array subscripts should be checked for out-of-bounds conditions. No special techniques were needed to implement this.

- Ability to handle necessary data types as an intrinsic part of the language, e.g., variable-length virtual memory string arrays and variable-length random access data files. The method of implementation of this feature is described later in this article.

- Ease of entering and changing programs by using a built-in word-processing-type editor.

- Portability of source code so that the same program will not have to be written over again for each type of computer.

- Ability to handle large memory configurations efficiently, without requiring the programmer to be concerned with memory allocation schemes.

- Encouragement of structured programming to reduce the likelihood of errors in the control flow.

- Support for long, mnemonic variable and subroutine names to improve readability.

- Support of BCD arithmetic to allow correct addition of totals in accounting reports. The OMNI commercial arithmetic package provides BCD add, subtract, multiply and divide in 16-digit accuracy.

- Support of slide-rule accuracy arithmetic having very high performance and great dynamic range.

It was necessary to write OMNI in assembly language to achieve maximum performance, which is especially important for providing a highly interactive environment as well as the computing resources necessary to support the language's other characteristics.

This may seem to contradict the ideal of portability, one of the characteristics of OMNI. Why didn't I write OMNI in C to gain portability?

The question has two answers. The first is that the user of a language is interested in the portability of the programs written in the language or those acquired from others. The user doesn't care how much work it is to transport the language from one machine to another since he or she doesn't have to do that.

The second reason is that no compiler in the world can generate code as well as an expert assembly language programmer. If OMNI were written in C, the OMNI compiler would probably run at one-tenth the speed it does now. Execution time efficiency would also be impaired.

The most difficult problem I had while writing OMNI was the implementation of the virtual memory. It had to be able to manage the storage and retrieval of strings and records, which could vary in length dynamically.

The design solution was to divide the mass storage into blocks of 1KB each, numbered from 1 to the maximum capacity of the system (limited to either 65,535 or approximately 2 billion blocks, depending on the processor). Each block is divided into up to 63 variable-length items, which are referred to by their relative item number. Therefore, each item in the virtual storage may be accessed by its block number and relative item number. Each block has a directory at its end, organized as shown in Table 2.

After the last item pointer and type, a dummy points to the free space in the block, which is always kept in a single segment. This allows the system to calculate the length of any item by subtracting its starting address from the starting address of the next item, with the free

Comparison of BASIC, COBOL, Forth, and OMNI

	Microsoft MBASIC		Microsoft COBOL v. 2.2		fig-Forth		OMNI	
	Rating	Value	Rating	Value	Rating	Value	Rating	Value
Compilation speed (Sieve, in sec)	4	<<1	0	146	3	2	4	<<1
Size of object code (Sieve, in bytes)	3	300	1	786	3	260	4	177
Execution speed (Sieve, in sec)	1	1,920	0	5,115	4	85	2	310
Variable name length	1	2 chars	4	31 chars	4	31 chars	4	31 chars
Local variables	0	none	0	none	0	none	3	scalars
Subroutine names	1	line ^	4	31 chars	4	31 chars	4	31 chars
Variable length strings	2	mem only	1	fixed	0	none	4	virtual
Variable length random-access records	0	none	2	inconvenient	0	none	4	virtual
Ease of extending language	1	assembly	1	assembly	4	high level	3	high level
BCD arithmetic	0	none	4	user-specific	0	none	3	fixed point
Engineering arithmetic (floating point)	3	accuracy	0	none	0	none	3	speed
Simple syntax	3	few rules	1	many rules	3	few rules	3	few rules
Meaningful error messages	3	in English	2	codes	1	often none	3	English
Interactive debugging	2	trace	0	batch	2	write own	4	single step
Run-time error checking	3	error msg	2	code	1	bomb	3	error msg
Source editor	1	line editor	2	screen	2	screen	4	word processing
Total	28		24		31		55	
Average	1.75		1.5		1.94		3.44	

0 is lowest, 4 is highest rating.

Sources: Gilbreath, John. "A High-Level Language Benchmark." *Byte* Sept. (1981): 180 and the author's experiments.

Table 1.

Byte Number	Size	Name in block
0	1	Block item count—tells how many variable length items this block currently contains
3,4	2	Same as above, but for second item in block
etc.		

Table 2.

space pointer serving this purpose for the last item in the block.

Retrieving the value of an item in the virtual memory was not particularly difficult to implement. For example, suppose that you ask for the value of the 300th element of string array #12. Of course, in a real program the array would have a name, but I am simplifying for purposes of discussion.

First OMNI looks up the pointer block number of array #12 in the master object table. Suppose the entry says "block 97, relative item #1". Block 97 is read into memory if it is not already there, and the

address of the first item in the block is looked up. Since string pointers are 3 bytes long, 300 is multiplied by 3 to yield an offset into the pointer array of 900.

At offset 900 in the array, a 3-byte string pointer is read. It says "block 102, item 27". So block 102 is read into memory, if necessary, and its item count is checked to see whether item 27 exists. If not, an error results. Assuming it does exist, its length will be calculated by subtracting its starting address from the starting address of the next item in the block, and it will be pushed onto the string stack.

This may seem like a lot of work just to retrieve the value of a variable-length string array element. But the reward is that the programmer no longer has to

worry about running out of RAM for such arrays. In OMNI you can create dozens or hundreds of arrays of thousands of variable-length string elements each, as long as your disk storage holds out.

The other unique problem that I encountered while writing OMNI was the requirement that definitions of operators be able to be removed from memory when no longer needed and their space reclaimed. This meant that all references to the address of an operator must be known to the system so that they could all be adjusted if a definition was removed or changed in length.

I wrote a subroutine called *MOVER*, which moved all definitions above the one deleted or changed and updated all entries in the definition address table and on the return stack, which was used to store return addresses during subroutine calls. This also meant that there must be no addresses in the OMNI object code, whether primitive or secondary. All primitives must use only position-independent addressing modes when referring to code or data contained in them, and all branches in secondaries must use relative addressing so that all operators will function correctly even if they are moved to different memory addresses.

As mentioned before, the OMNI compiler is fast for three main reasons. First, since the language uses reverse polish notation, the compiler doesn't have much work to do compared to the code needed to unravel complicated hierarchical expressions. Also, parameter passing is done mainly on the stack, which reduces subroutine linkage complexity.

Second, the look-up of indexes of operations, given their names, is speeded up immensely (a factor of 10 or more, depending on the number of operations currently defined) by using hash coding.

Third, the code necessary to handle compilation of such things as loops, conditional execution, and variable definitions is distributed to the operators responsible for those specialized tasks and not centralized in the compiler. This allows extensions to the language without continually increasing the size and complexity of the main compiler, which handles only operator name look-up and conversion of numeric constants from ASCII to internal format.

The main tool for debugging OMNI programs is the tracing inner interpreter (II). The II controls the execution of applications programs in OMNI and can be switched from one mode to another by changing one pointer.

GOOD NEWS!



C for the 6809 WAS NEVER BETTER!

INTROL-C/6809, Version 1.5

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

I wrote a specialized II that displays the name of the next operation to be performed (if it has one) and the current contents of the stacks before it executes each operation. It then waits for a keystroke, after which the II accepts a line of code that can contain almost any legal OMNI code.

This same code will then be compiled and executed. This allows you to watch the execution of an operator and interrupt it at any point, change the stack contents, or display or modify the value of a variable, and then continue where you left off. Since a large percentage of program errors in a reverse polish notation language are caused by incorrect stack usage, the constant display of the stacks is very useful in debugging.

The word processor can be run in "hot" mode, which allows immediate compilation of a section of the source program currently being edited and the testing of it, without ever leaving the word processor. This allows even greater interactivity than would be assumed from the rating of compiler speed in Table 1 since it takes only a fraction of a second to recompile a section of program of 20 lines or so, which covers the vast majority of operators.

OMNI attempts to give as much protection against programming errors as possible. Because you cannot address memory directly, but only through carefully controlled and monitored access operators, it is essentially impossible to corrupt the object code of your program in memory or store into a data area that is not appropriate.

Memory is viewed only in the guise of variables and arrays. Subscripts are checked at run time since one of the most common programming errors is to violate the boundaries of an array and destroy something else in memory. If a subscript error is detected, an error is raised. This also applies to BCD overflow, which is generally a fatal error anyway.

When an error occurs, the compiler or interpreter indicates what the subroutines were that were suspended at the time of the error in an upward order from the lowest one.

Because the edit/compile/test cycle is so rapid, I have not felt it necessary for compilation to continue after the first error is detected and announced. This may be changed in the future.

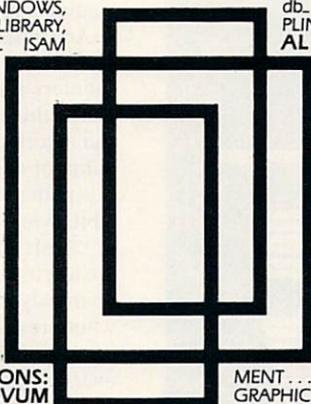
Portability to me means that the same program should produce the same results on any machine on which it will run at all, and it will run on more than one machine of differing architecture. This is a major goal of OMNI.

Portability affects the implementation of OMNI in a variety of ways, which can be summarized as follows:

Once you choose Lattice, our friends will C you through...

LATTICE INC.: LATTICE WINDOWS, CURSES UNIX SCREEN CONTROL LIBRARY, C-FOOD SMORGASBORD, dB-C ISAM COMPATIBLE WITH dBASE II AND III...

LIFEBOAT ASSOCIATES: FLOAT 87 8087 SUPPORT PACKAGE, HALO GRAPHICS PACKAGE, PANEL SCREEN LIBRARY... **GREENLEAF SOFTWARE:** THE GREENLEAF C FUNCTIONS... **C SOURCE:** BASIC_C C FUNCTIONS FOR BASIC USER... **SOFTCRAFT:** BTRIEVE ISAM FILE SYSTEM, BTRIEVE ISAM NETWORK FILE SYSTEM... **BLAISE COMPUTING:** TOOLS, TOOLS2, VIEW MANAGER SCREEN PACKAGE... **MORNING STAR SYSTEMS:** PROLIBRARY, PROSCREEN... **CREATIVE SOLUTIONS:** WINDOWS FOR C... **NOVUM ORGANUM:** C POWERS PACKS, MATH-EMATICS POWER PACKS, ADVANCED POWER PACKS, DATABASE POWER PACKS, TELECOMMUNICATIONS POWER PACKS W/ SOURCE... **PHACT ASSOCIATES:** PHACT ISAM LIBRARY... **RAIMA CORPORATION:**



db_ VISTA DBMS... **PHOENIX:** PLINK86, PFIX86... **RELATIONAL DATABASE SYSTEMS:** C-ISAM FILE ACCESS METHOD... **MINDBANK:** V-FILE VIRTUAL MEMORY/FILE SYSTEM... **HUNTER & READY:** VRTX C INTERFACE LIBRARY... **GRAPHIC SOFTWARE SYSTEMS:** GSS DRIVERS, GSS TOOLKIT KERNEL SYSTEM... **OPT-TECH DATA PROCESSING:** OPT-TECH SORT... **ACCUDATA SOFTWARE:** C-TREE ISAM, C-SORT SORT... **TRIO SYSTEMS:** C-INDEX+ ISAM... **COMPU CRAFT:** C VIEW FORMS/WINDOW MANAGEMENT... **SCIENTIFIC ENDEAVORS:** GRAPHIC PRESENTATION SCIENTIFIC GRAPHICS... **LEMMA SYSTEMS, INC.:** C LIBRARY... **ESSENTIAL SOFTWARE, INC.:** C UTILITY LIBRARY... **SOFTWARE LABS:** C UTILITIES PACKAGE... **FAIRCOM:** C-tree BY FAIRCOM ISAM WITH SOURCE

Contact Lattice to learn how we can help your C program development.



LATTICE®

P.O. Box 3072
Glen Ellyn, IL 60138
312/858-7950
TWX 910-291-2190

CIRCLE 36 ON READER SERVICE CARD

QUALITY SOFTWARE AT REASONABLE PRICES

CP/M Software by

Poor Person Software

Poor Person's Spooler **\$49.95**

All the function of a hardware print buffer at a fraction of the cost. Keyboard control. Spools and prints simultaneously.

Poor Person's Spread Sheet **\$29.95**

Flexible screen formats and BASIC-like language. Preprogrammed applications include Real Estate Evaluation.

Poor Person's Spelling Checker **\$29.95**

Simple and fast! 33,000 word dictionary. Checks any CP/M text file.

aMAZEing Game **\$29.95**

Arcade action for CP/M! Evade goblins and collect treasure.

Crossword Game **\$39.95**

Teach spelling and build vocabulary. Fun and challenging.

Mailing Label Printer **\$29.95**

Select and print labels in many formats.

Window System **\$29.95**

Application control of independent virtual screens.

All products require 56k CP/M 2.2 and are available on 8" IBM and 5" Northstar formats, other 5" formats add \$5 handling charge. California residents include sales tax.

Poor Person Software

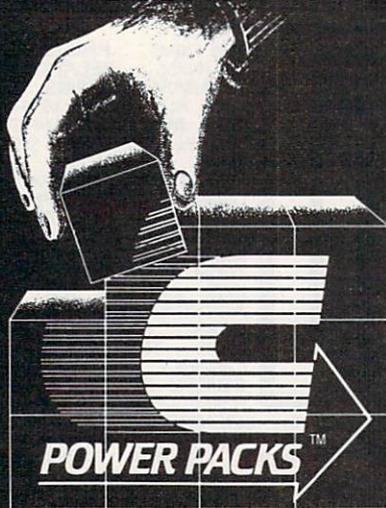
3721 Starr King Circle
Palo Alto, CA 94306
tel 415-493-3735

CP/M is a registered trademark of Digital Research

CIRCLE 51 ON READER SERVICE CARD

"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, Async
- **PACK 2: Database** \$399
100 Functions: B-Trees, Variable Records
- **PACK 3: Communications** \$149
135 Functions: Smart-modem™, 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%)



165 Bedford Street
Burlington, Mass. 01803
(617) 273-4711

NOVUM ORGANUM

CIRCLE 25 ON READER SERVICE CARD

■ **Memory and I/O control.** No application program may access memory or I/O devices except through the intermediary of OMNI primitives. Memory may be viewed only as variables and arrays, and I/O devices are treated as files, with standard conventions for opening, closing, reading and writing.

■ **Arithmetic operations.** All OMNI systems use 16-bit integer arithmetic for counters and similar uses, 16-digit decimal arithmetic for commercial arithmetic and record numbers, and 16-bit floating point for engineering applications.

Another of the great drawbacks to portability in C is the fact that the precisions of "short" and "long" integers and reals are left to the implementor to decide. This obviously allows the writing of programs whose results depend on the actual length of such numbers, especially if arithmetic overflow is not detected at run time.

■ **File access.** OMNI files can have up to about 2 billion records, each of which can have up to approximately 32,000 fields. The total mass storage supported as one virtual address space varies from system to system since 32-bit arithmetic is unreasonably slow and large on some 8-bit computers, such as the Z80. On those machines, the virtual address space is limited to approximately 65MB. On other machines, such as in the 68000 or 8086 families, the limit is approximately 2 trillion bytes!

In any discussion of software portability, someone will express the opinion that portability is unnecessary because all the software developer needs to do is be IBM PC compatible. This is an attractive notion as there are always trade-offs in attempting to write portable code.

However, I believe this idea to be erroneous for the following reason: IBM compatibility cannot be attained because even the various models of the IBM PC line are not hardware compatible. Leaving aside the PCjr, which is not a business-oriented machine, and the PC 370XT, which is designed to run IBM System 370 software, the new PC AT is different architecturally from the PC and PC XT as it uses an 80286 processor rather than the 8088 that the other two use.

Since memory segmentation works differently on the 80286 than it does on the 8088, programs that modify the segmentation registers are not transferable between these processors without major changes. This includes all major business programs for the IBM PC line, such as Lotus 1-2-3. Even worse, IBM appears to have deliberately prevented the "compatible" machines from running IBM's new software.

The only solution to this problem is for

developers to use a virtual machine such as OMNI. Then a new machine will necessitate the rewriting of only one program, the OMNI system. The developers will be able to go on to write new programs rather than struggle to keep up with IBM.

As previously mentioned, the size of OMNI object code is extremely small—less than 2 bytes per use of each operator on average. Overhead for definitions of operators is also small—it amounts to 5 bytes plus the name of the operator.

The names of any or all operators can be removed from the object code at any time and the space thus saved reclaimed. This adds to program security since an operator whose name has been removed can no longer be compiled into a new definition and further reduces space requirements. (For example, source code for the Sieve of Eratosthenes, excluding comments, is 487 bytes and the object code is 177 bytes.)

A final point I'd like to make is that a great deal of room exists for disagreement on the readability of various programming languages. It is as easy to write unreadable programs in OMNI as it is in APL and Forth.

However, it is also easy to write readable ones. You must make liberal use of variables and restrict to the minimum your use of the stacks. In fact, it is not necessary to use the stacks explicitly at all (although performance may be improved if you do use them). Take the following examples of what to do and what not to do:

(bad) $ABC \text{ over over } + * \text{ swap } -$
 $+ - > D$
(good) $CC * BC * + B - A +$
 $- > D$
(The BASIC equivalent would be: $D = C * C + B * C - B + A$)

If you'd like to learn more about OMNI, leave me a message on the Bulletin Board Service or send me a note c/o *COMPUTER LANGUAGE*, 131 Townsend St., San Francisco, Calif. 94107. Tell me what you think about the design features I developed with OMNI and ways you think it might be improved. ■

Steve Heller graduated from Shimer College around 1970 and became a programmer more or less by accident. He has been programming for the past 15 years.

A conversation with CompuPro's Bill Godbout

By Regina Starr Ridley

"How're you doing, Bill!" a young CompuPro employee greets Bill Godbout as he walks through a large warehouse room where 20 or so employees are eating lunch, playing ping pong, or working amid an amazing jumble of hardware.

Godbout, chairman and CEO of CompuPro, responds warmly. He wears a blue plastic name tag that says Bill Godbout, probably put on absentmindedly, out of habit. Everyone here knows who Bill is. And you get the feeling everyone likes this big, friendly man who looks like he'd make an excellent Santa Claus.

Godbout is an entrepreneur—a very successful one whose 25 years or so in business appear unblemished with failure. But he's no cutthroat business man. He talks about CompuPro as a family and his life seems to have been filled with varied, unusual experiences, many of which Godbout is pleased to share. He loves to tell a good anecdote, joke, or bit of trivia. And he knows plenty of them.

Originally from the East Coast, Godbout came west to California's Bay Area after being on active duty in the military for most of the 1960s. He had been involuntarily recalled to active duty in 1961—the Berlin Wall period—while working for IBM.

Godbout put in for duty in the Far East hoping to go to Korea, which he heard was the best kept secret in the army, or Japan, with its geisha houses and the Aki Habara, an electronics trade zone. Shortly before Christmas of 1961 he got orders to prepare for a transfer and, as he puts it, he thought "ah! I've made it! Geisha houses, the Aki Habara . . ."

"I went to bed that night happy as a little tick," Godbout says smiling, shaking his head. "Little did I know . . ."

Godbout was discharged in 1968. Although he held IBM in esteem, he decided he would not to work for a big government or a big company again. But, he said, "If I were to work for a big company I sure as hell would consider IBM right up at the top of the list."

Godbout moved to the Bay Area because someone he knew was putting together a crisis management team to help save a company in deep financial trouble

and asked him to join. The team had 18 months to turn the company around—they did it in 13.

The team members figured they had done so well they might as well start their own company. But they quickly found out that knowing how to save a foundering company is not the same as making a success of a new one. "We did all the same wrong things that those guys had done to get into trouble except for one thing," said Godbout. "Once we really got into difficulties we knew what to do. We could work our way out."

"It was a very, very low profile company," said Godbout, dropping his voice to a conspiratorial whisper and declining to reveal the company's products. The kicker was that the company was located in Oakland, the town right next to Berkeley—not exactly a peaceful area during the late sixties and early seventies.

"We had visions of people burning cars in the parking lot," said Godbout. But the low profile was mostly successful except for one frustration.

"It wasn't like James Bond and that was a big disappointment," Godbout said. "Never once in all those years did a gorgeous anything—blonde, redhead, or brunette—ever try to seduce me. I could have been had too! I should have been in the FBI. I guess that's the way to get the action."

Godbout sold the business and went into a period of semi-retirement, "retirement on the installment plan." It was a time, he said, of bumming around, fondling airplanes, playing at the airport, drinking a lot, and "doing the good old boy thing till I was bored to death."

Then came the beginnings of CompuPro.

A good friend of Godbout's, Mike Quinn, had a place at the Oakland airport where he sold surplus semiconductors and miscellaneous electronic spare parts. Godbout was fascinated.

But maybe the true start of CompuPro was when Godbout was a little boy. "To drop back many years. I guess really all my life I've been a dump picker at heart. As a kid with a paper route I used to pass a dump behind a GE plant. There were lots of wonderful and useful things back there,



like meters. It might have a bent needle but you could straighten it out."

"When I look back at the amount of time it took to take the thing apart and then really get the damn thing working, considering the time factor, it would cost a technician twice as much. But at that time, from my point of view, I thought, 'Oh God, what's the matter with these people?' I would drag these things home, much to the chagrin of my mother."

Godbout's friend was buying electronic parts that semiconductor firms couldn't economically deal with and were selling at upset prices. Parts also came from companies going out of business. The real job was redistributing the parts into markets which had been unreachable. This too reminded Godbout of his boyhood.

"I recall the first transistor I got when I was a kid. It was treated like a crown jewel," said Godbout. It was called a CK722 and was the first commercially made transistor.

"God I don't recall how much paper route money it took to obtain that damned thing but I managed to get one. I got it from a wholesaler who I think took pity on me. He sold me just one unit at the one-thousand-piece price because this kid showed up with money clutched in hand to see if he could get a transistor."

"I could see people like me of all ages who were interested in electronics. And you would see these neat projects in *Popular Electronics* and *Radio Electronics* magazines, and you'd have to scrounge like mad to find the parts to build the damn things."

First Godbout became interested in the acquisition of surplus parts and spent

Multi-Basic

"The BASIC compiler that compiles both MBASIC and CBASIC"

Now you don't have to give up the features you like about MBASIC to obtain the powerful capabilities of CBASIC. Multi-Basic gives you both.

Multi-Basic works with your existing programs so your current software investment is protected. But just as important, Multi-Basic opens the door to a whole new way of programming. With Multi-Basic you can write very readable, modular and structured programs. Multi-Basic makes program maintenance as easy as it is with Pascal.

In addition to understanding the two most popular dialects of BASIC, Multi-Basic allows you to extend the language even further. You can add your own statements and functions as needed.

Multi-Basic is also compatible with our Pascal and C compilers. This allows your BASIC programs to use routines written in Pascal or C.

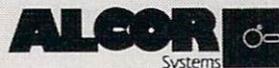
In today's fast changing computer business, you need a language as versatile as Multi-Basic. Invest a little time today and save a lot of time tomorrow. You owe it to yourself to see what a difference Multi-Basic can make.

Multi-Basic is available for the TRS80 models I, II, III, 4 and 12; Tandy 2000, IBM PC, and CP/M. It is compatible with TRSDOS, LDOS, NEWDOS, DOSPLUS, MSDOS, PCDOS, CP/M and CP/M plus.

Alcor Multi-Basic \$139

Other Products:

Advanced Development Package	\$ 69
Blaise I Text Editor (Mod 1 or 3)	\$ 49
Blaise II Text Editor (all others)	\$ 79
Multiprocessor Assembler	\$ 69
Alcor C	\$139
Alcor Pascal (for CP/M, MSDOS, PCDOS)	\$139
Complete Development System	\$250
includes compiler, text editor and advanced development package	
Shipping U.S.A.	\$6.00
Shipping Overseas	\$28.00



13534 Preston Road, Suite 365
Dallas, Texas 75240
(214) 494-1316

Multi-Basic is a trademark of Alcor Systems
TRS80 is a registered trademark of Tandy Corporation
CP/M, CBASIC are trademarks of Digital Research
MSDOS, MBASIC are trademarks of Microsoft

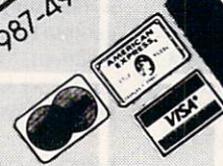
CIRCLE 1 ON READER SERVICE CARD

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...

\$299.95

FOR MORE INFORMATION OR TO ORDER CALL OR WRITE:
Megamax, Inc.
(214) 987-4931



MACINTOSH IS A REGISTERED TRADEMARK OF APPLE COMPUTER INC.

NEW FOR THE MACINTOSH

CIRCLE 13 ON READER SERVICE CARD

Fortran Scientific Subroutine Package

Contains Approx. 100 Fortran Subroutines Covering:

- | | |
|----------------------------------|-----------------------------|
| 1. Matrix Storage and Operations | 7. Time Series |
| 2. Correlation and Regression | 8. Nonparametric Statistics |
| 3. Design Analysis | 9. Distribution Functions |
| 4. Discriminant Analysis | 10. Linear Analysis |
| 5. Factor Analysis | 11. Polynomial Solutions |
| 6. Eigen Analysis | 12. Data Screening |

Sources Included, Microsoft 3.2 compatible.
\$295.00

FORLIB-PLUS™

Contains three assembly coded LIBRARIES plus support, FORTRAN coded subroutines and DEMO programs.

The three LIBRARIES contain support for GRAPHICS, COMMUNICATION, and FILE HANDLING/DISK SUPPORT. An additional feature within the graphics library is the capability of one fortran program calling another and passing data to it. Within the communication library, there are routines which will permit interrupt driven, buffered data to be received. With this capability, 9600 BAUD communication is possible. The file handling library contains all the required software to be DOS 3.0 PATHNAME compatible.

\$69.95
Strings & Things™
Character Manipulation and Much More!
\$69.95



P.O. Box 2517
Cypress, CA 90630 (714) 894-6808

California residents, please add 6% sales tax

CIRCLE 2 ON READER SERVICE CARD

some time as Quinn's apprentice. On the business level he found it intriguing as a case study on how companies dispose of their excess inventory. On another level, the quasi-hobbyist or hacker in him thought, "gee, here's a neat treasure to be shared."

So in the early seventies Godbout started a mail order business that was strictly parts. In 1973 he started playing around with the new microcomputer equipment and then made a deliberate attempt to gather different pieces of computer parts and offer them for sale. The first computer kit was done in September 1973 and went over very well. He also got into music kits. These two business angles later became separate divisions: CompuKit and MusiKit.

Godbout started catering to the small groups of people who were actually building their own machines. CompuKit was the first company to nationally distribute a semiconductor memory kit, said Godbout.

The kit was pre-Altair and "a marvel of its time," said Godbout, with proper awe in his voice. "It was a 4K byte board about 1-foot square and just teaming with power hungry parts. Nowadays, using the same technology, we could easily stick a megabyte on the same size board and consume a hell of a lot less power."

CompuPro has come quite a long way in the 11 years following the first computer kits. The company primarily became known as a leading supplier of S-100 micro-computer products used to integrate packaged systems.

In 1982 the company introduced the System 816 series, its first fully integrated business systems. A year or so later, CompuPro introduced the CompuPro 10, the first product that was a real departure from the company's usual S-100-oriented products.

The System 816 series and the CompuPro 10 are considered two subsets of the same family and are referred to as the letter series and the number series, respectively.

The System 816 series is based on microprocessors from four manufacturers—Intel, Motorola, National Semiconductor, and Zilog—and distinguished with a slash and letter following the 816.

The number series, though conceptually single-board machines, are essentially fixed architecture with limited growth or modification potential, said Godbout. They are not bus-based—they have expansion slots. The CompuPro 10 has six slots.

The CompuPro 10 is aimed at the business office environment. In contrast, a large percentage of the S-100 market business is actually OEM business, which Godbout estimates to be more than 50% of the product manufactured. He can cite

numerous large companies that use CompuPro S-100s.

The introduction of the CompuPro 10 has not been without a number of problems, Godbout states openly. The 10 has a Master-Slave architecture where a central 16-bit, 8-MHz 8088 processor is dedicated to handle system resources and four 8-bit, 6-MHz Z80B processors act as individual users.

"We had test problems, we had code problems, and we ended up, albeit very quietly, putting binders on shipments after the initial introduction. We froze shipments and we pulled back all the machines and exchanged them. With everyone we could identify we just shipped them a machine and said send the old one back. If they called up and said, my hortneimer valve doesn't ragfrazzle, we said please put that machine in a box and send it back to us collect."

The 10 had a number of problems. "And we had a number of false starts. We thought we had the problem pegged—we were doing field tests and beta tests and running everything parallel at top speed to catch these things. But it was a matter of months till we really got the 10 settled down and stabilized."

All the machines that have been shipped since May of this year have been stable and no changes have been necessary since then, said Godbout. CompuPro is now effecting one planned change—moving the operating system from Digital Research's MP/M to Concurrent. CompuPro 10s shipped in October have the Concurrent operating system.

CompuPro has stayed with the CP/M products rather than follow the PC-DOS path, and Godbout is quite a supporter of CP/M.

"I look upon CP/M and the CP/M family like Listerine. You may not be in love with it but it's the best damn game in town."

"I think DRI products are clearly superior products. MP/M 86, CP/M 86, and Concurrent are true operating systems," said Godbout. "They provide a real interface that makes for true hardware independence. You can't really say that for any other operating systems. I can't think of anything that approaches Concurrent and its successors."

"I'm personally convinced that from an applications programmer's standpoint, and from a hardware designer's standpoint, you can do a lot more, better, in a Concurrent CP/M environment than any other environment. It is a general purpose operating system. The market we're dealing with is a general purpose market."

But Godbout considers the future of CP/M "iffy." As he says, "it wouldn't be the the first time that a really technically superior product has gone down the drain in the face of a really less meritorious product. But I don't think DRI's demise is imminent. They just have a big marketing task ahead of them."

NEW FEATURES

(Free update for our early customers!)

- Edit & Load multiple memory resident files.
- Complete 8087 assembler mnemonics.
- High level 8087 support. Full range transcendental (tan, sin, cos, arctan, logs and exponentials) Data type conversion and I/O formatting.
- High level interrupt support. Execute Forth words from within machine code primitives.
- 80186 Assembler extensions for Tandy 2000, etc.
- Video/Graphics interface for Data General Desktop Model 10

HS / FORTH

- Fully Optimized & Tested for:
IBM-PC IBM-XT IBM-JR
COMPAQ EAGLE-PC-2
TANDY 2000 CORONA
LEADING EDGE

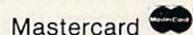
(Identical version runs on almost all MSDOS compatibles!)

- Graphics & Text (including windowed scrolling)
- Music - foreground and background includes multi-tasking example
- Includes Forth-79 and Forth-83
- File and/or Screen interfaces
- Segment Management Support
- Full megabyte - programs or data
- Complete Assembler (interactive, easy to use & learn)
- Compare
BYTE Sieve Benchmark jan 83
HS/FORTH 47 sec BASIC 2000 sec
W/AUTO-OPT 9 sec Assembler 5 sec
other Forths (mostly 64k) 70-140 sec
FASTEST FORTH SYSTEM AVAILABLE.

TWICE AS FAST AS OTHER FULL MEGABYTE FORTHS!

(TEN TIMES FASTER WHEN USING AUTO-OPT!)

HS/FORTH, complete system only: \$250.

 Visa  Mastercard

Add \$10. shipping and handling

HARVARD SOFTWARES

PO BOX 2579
SPRINGFIELD, OH 45501
(513) 390-2087

CIRCLE 47 ON READER SERVICE CARD

C

SOFTWARE DEVELOPERS!

V - FILE THE VIRTUAL MEMORY FILE MANAGER

Let V-FILE save precious development time & cost as you create efficient applications with the power of VIRTUAL MEMORY.

DON'T RE-INVENT THE WHEEL

Why spend weeks or months coding and debugging file and memory management systems when you can order V-FILE today. V-FILE is a library that you can link with your code to provide sophisticated virtual file and memory management — allowing you to concentrate on developing your application.

VIRTUAL DATA OBJECTS SUPPORTED!

Data is referenced by using VIRTUAL MEMORY DATA HANDLES. Your code doesn't need to know whether the data is actually on disk or in RAM. Swapping between disk and RAM and updating files on disk is handled automatically and transparently! Complex VIRTUAL DATA STRUCTURES can be created by linking with data handles instead of pointers.

CHECK THESE FEATURES!

- Multiple, independent swap buffers
- Multiple files per swap buffer
- Highly efficient swap algorithm
- Automatic file updating
- Data prefetching supported
- Data may be locked in memory
- Memory buffers may be flushed
- Makes full use of extended memory on IBM PC/AT
- SOURCE CODE AVAILABLE
- NO ROYALTIES REQUIRED

Supports Dos 2.00+ with
Lattice & Microsoft C compilers
Supports Microsoft windows



TM
VISA/MASTER CARD ACCEPTED

\$299

Contact:
MindBank, Inc.
4620 Henry Street
Pittsburgh, PA 15213
412/683-9800

CIRCLE 63 ON READER SERVICE CARD

Godbout categorized PC-DOS as the approximate equivalent of CP/M 2.2—a job control language. "At the risk of sounding churlish or foolish," said Godbout, "IBM has not been noted for the technical excellence of its hardware or software.

"You won't see the 10 running IBM PC-DOS software," said Godbout. But CompuPro purchased the license for PC compatibility along with Concurrent and will be shipping the PC compatibility module with every Concurrent operating system for bus-based machines.

"I don't care what you call the operating system itself—what you're going to end up with as I see it is a framework, a hierarchical structure that will provide for calls for a number of job control languages or subset modules to interface with things like PC-DOS, the older CP/M, the new CP/M, and even a UNIX-like operating system. And the appropriate one will be called when you load a program."

Godbout believes that the point is just now being reached where a realistic piece of hardware is available for doing this type of thing. He feels CompuPro's System 286, based on Intel's iapx 80286, is an example of such a machine that permits very rapid context swaps without a tremendous amount of software or operating systems or systems programmer manipulation or overhead. The 286 hardware has backup to support software partitioning between users or tasks and provides four levels of privileged access.

The CPU 286 was introduced in November 1982 at Comdex. The 286s were tough to come by last year, said Godbout, and they were high priced machines—\$15,000 to \$20,000 without a terminal or printer.

Although it has not yet been announced, Godbout said that CompuPro now has a UNIX System V running on the 32016. It's being tested in Europe and, as soon as AT&T "anoints it" and the System V is released, CompuPro will have a System V on the street for the 32016.

Godbout says the product is a true 32-bit processor that fits on an S-100 bus. "The 32016 bus is a 32-bit guy that's double pipelined and has both an instruction queue and memory address processor queue."

The 286 was designed from the ground up for multi-using and multitasking and has the fast context swap instruction plus instructions like popa and pusha, which make for very fast changes of personality. (The "a" on the end stands for all—pusha will take all of the stack and move it with just that one instruction.) The 286 also has an onboard memory manager that provides for up to a gigabyte of virtual storage space per user. So big virtual machines can be built in the operating system and you get a hardware base to execute it with, Godbout said.

The 32016 requires an external memory manager. It has virtual memory but its instruction set is very orthogonal.

It's more suited to the UNIX-type environment than some of the other machines, he said.

Now that CompuPro has branched off into a very different market area, which market will it push more strongly?

CompuPro is coming down with feet planted on both sides of the fence, according to Godbout. The CompuPro 286, which is bus based, was to be introduced at Comdex in November, so CompuPro is not ignoring the bus machines, he said. And to go to the 32-bit and bigger machines CompuPro will have to go to a new bus. A final decision has not yet been made except that a new bus will definitely be added to CompuPro's bus line.

Godbout considers the development systems to be solid business and doesn't want to abandon or neglect this area at all. But he expects that CompuPro will experience its largest growth in the next three to five years in the office automation area.

Selling in the competitive business market requires a strong marketing program. In addition to this new area, CompuPro plans to pursue other markets, including artificial intelligence. For example, CompuPro now has LISP for the 68K system, the Cambridge LISP. CompuPro Europe has released UNIX for the 68K system—it was tested in the U.K., is now in Canada, and will be showing up in the U.S.

The name CompuPro did not seem adequate to appeal to these new markets and to cover a diverse product mix. Marketing tests showed that many people found it hard to pronounce and weren't able to relate the written and the spoken word.

So CompuPro went to Namelab, a company that thinks up new names, early this year. Eight possible new names have been chosen and are being run through trademark searches and word checks in other languages.

CompuPro would like to announce the new name January 1. The number one contender, says Godbout, is nice, euphonious, different, and in 9,999 times out of 10,000 times it is pronounced correctly when read. It is three syllables long, has more than five letters, and doesn't sound high-techy like Exxon.

"We had to have a name that would be totally neutral or Pablum—a standard brand," said Godbout.

It's hard to imagine this witty man, never without a sparkle in his eye, helping to come up with a Pablum name. A real zinger would probably be a better bet.

Another good bet would be to go with this entrepreneur, who claims there is no secret to success. "In all endeavors you've just got to do it. You give it your best shot and you get it out the door."

Regina Starr Ridley is managing editor of COMPUTER LANGUAGE.



Program Editing is finally both: Intuitive and Powerful ... and configurable to suit your style

BRIEF lets you concentrate on programming by keeping the Editor "out of the way," while combining power and natural flow:

- Full UNDO (N Times)
- Edit Multiple Large Files
- True Automatic Indent for C
- Exit to DOS Inside BRIEF
- Uses All Available Memory
- Intuitive Commands
- Tutorial
- Repeat Keystroke Sequences
- Windows (Tiled and "Pop Up")
- Unlimited File Size
- Reconfigurable Keyboard
- Online Help
- Search for Complex Patterns
- Mnemonic Key Assignments
- Horizontal Scrolling
- Comprehensive Error Recovery

PLUS a Complete, Powerful, Readable, Compiled MACRO Language

Availability: PCDOS-compatible systems. Price: Only \$195.

Win \$1,000 and recognition for the Outstanding Practical BRIEF Macro. Other awards to be given.

Try BRIEF. Use the Demo ... or the full product for 30 days. Call or write us ... 617-659-1571

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

Solution Systems™

335-L Washington St., Norwell, MA 02061

CIRCLE 71 ON READER SERVICE CARD

WALTZ LISP (TM)

The one and only **adult** Lisp system for CP/M users.

Waltz Lisp is a very powerful and complete implementation of the Lisp programming language. It includes features previously available only in large Lisp systems. In fact, Waltz is substantially compatible with Franz (the Lisp running under Unix), and is similar to MacLisp. Waltz is perfect for Artificial Intelligence programming. It is also most suitable for general applications.

Much faster than other microcomputer Lisps. • Long integers (up to 611 digits). Selectable radix • True dynamic character strings. Full string operations including fast matching/extraction. • Flexibly implemented random file access. • Binary files. • Standard CP/M devices. • Access to disk directories. • Functions of type lambda (expr), nlambda (fexpr), lexpr, macro. • Splicing and non-splicing character macros. • User control over all aspects of the interpreter. • Built-in prettyprinting and formatting facilities. • Complete set of error handling and debugging functions including user programmable processing of undefined function references. • Virtual function definitions. • Optional automatic loading of initialization file. • Powerful CP/M command line parsing. • Fast sorting/merging using user defined comparison predicates. • Full suite of mapping functions, iterators, etc. • Assembly language interface. • Over 250 functions in total. • The best documentation ever produced for a micro Lisp (300+ full size pages, hundreds of illustrative examples).

Waltz Lisp requires CP/M 2.2, Z80 and 48K RAM (more recommended). All common 5" and 8" disk formats available.

PRO CODE (TM)

INTERNATIONAL

Version 4.4

\$169*

(Now includes Tiny Prolog written in Waltz Lisp.)

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

Call free **1-800-LIP-4000** Dept. #13
In Oregon and outside USA call 1-503-684-3000

CIRCLE 53 ON READER SERVICE CARD

ACTIVE TRACE

"Software that lives up to its promises. When a Basic program doesn't work the way you want it to, this package... will help you track the problem down... Scope is a tool for the beginning, advanced, or professional programmer, and it begins where the cross reference maps leave off."

Howard Glosser, *Softalk for the IBM Personal Computer*
July '84, pp 120-121

"Extremely useful program..."

Anyone doing much programming in Basic should appreciate Active Trace a lot."

Jerry Pournelle, *Byte Magazine*
April '83, p 234

"A marvelous Basic programming aid... It's just amazing to watch a program you wrote run under Scope, and debugging becomes if not trivial, then at least doable"

Thomas Bonoma, *Microcomputing*,
Dec. '83, p 22

"... a really neat utility... designed to untangle even the most convoluted Basic program... The documentation is almost worth the price of the package."

Susan Glinert-Cole, *Creative Computing*, July '84, p 210

Active Trace will lead you through your program letting you know variable values (all variables or just those you specify) as they change. Your program's internal activity is presented on your screen, or printer, or it can be saved on disk. It's simple, effective and works with the BASIC you already own.

Active Trace \$79.95
Includes Scope, XREF mapping and documentation

Active Trace is available for most MS-DOS and CPM 2.2 systems and supports the special features of Brand specific versions of Microsoft Basic such as **Basica** on the IBM-PC.

AWARECO

Active Software

P.O. Box 695 Gualala, CA 95445
(707) 884-4019
800-358-9120(US) 800-862-4948(CA)

Active trace, Active software, and Scope are trademarks of AWARECO—CPM is a trademark of Digital Research—MS-DOS and Microsoft are trademarks of Microsoft Corporation—IBM-PC is a trademark of IBM Corp.

CIRCLE 3 ON READER SERVICE CARD

Announcing a

TOTAL PARSER GENERATOR

< GOAL > :: = < RAPID > < COMPILER > < DESIGN >

SLICE YOUR COMPILER DEVELOPMENT TIME

An LR(1) parser generator and several sample compilers, all in Pascal for your microcomputer.

- Generates parser, lexical analyzer and skeleton semantics
- Universal, state-of-the-art error recovery system
- Adaptable to other languages
- Interactive debugging support
- Thorough documentation
- **TURBO PASCAL™ INCLUDED FREE OF CHARGE**
- Includes mini-Pascal compiler, assembler, simulator in SOURCE

SPECIAL INTRODUCTORY OFFER \$1995

OPARSER™ runs on IBM PC/DOS in Turbo Pascal. Parser generator in object form, all else in source. OPARSER takes a grammar and generates a correct, complete, high-performance compiler with skeleton semantics in Pascal source. Easy to add full semantics for YOUR application. Excellent for industrial and academic use. An accompanying textbook (SRA publishers) available in 1985. Training can be arranged. Demo disk available for \$50.

Educational and quantity discounts available. Check, money order, Mastercard, Visa. California residents add 6.5% sales tax.

WRITE OR CALL FOR FREE BROCHURE

Technical details: call 408/255-5574. Immediate delivery. CALL TODAY!

QCAD

SYSTEMS, INC.

1164 Hyde Ave., San Jose, CA 95129

TOLL FREE: 800-538-9787
(California residents call 408/255-5574)

™ Turbo Pascal is a registered trademark of Borland International.

CIRCLE 23 ON READER SERVICE CARD

LOWER PROGRAMMING MAINTENANCE AND DEVELOPMENT COSTS

{SET:SCIL}

The Source Code Interactive Librarian for microcomputers.

- SCIL keeps a historical record of all changes made to the library.
- SCIL maintains any source code regardless of language, including user documentation and text material.
- SCIL allows software engineers to work with source code as they do now, *using any ASCII text editor.*
- SCIL saves disk space by storing only the changes made to the program.
- SCIL provides a labeling capability for ease of maintaining multiple versions and multiple releases.
- SCIL offers unlimited description in the program library directory.
- *High visibility displays* with varied intensity for ease of viewing insertions and deletions.
- SCIL is available on CP/M, MP/MII, MS-DOS, PC-DOS and TurboDOS.

{SET} Get {SET} for Success
{SET:SCIL}™ is a product of System Engineering Tools, Inc.
645 Arroyo Drive, San Diego, CA 92103

Registered Trademarks: CP/M, MP/MII, Digital Research, Inc., MS-DOS, Microsoft Corp., PC-DOS, IBM Corp., TurboDOS, Software 2000, Inc.

For more information call (619) 692-9464.

CIRCLE 64 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

SUPER FORTH 64™ TOTAL CONTROL OVER YOUR COMMODORE-64™ USING ONLY WORDS MAKING PROGRAMMING FAST, FUN AND EASY!

MORE THAN JUST A LANGUAGE...

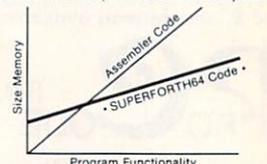
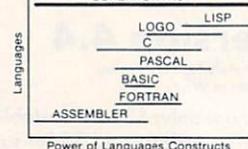
A complete, fully-integrated program development system.

Home Use, Fast Games, Graphics, Data Acquisition, Business Real Time Process Control, Communications, Robotics, Scientific, Artificial Intelligence

A Powerful Superset of MVPFORTH/FORTH 79 + Ext. for the beginner or professional

- 20 to 600 x faster than Basic
- 1/4 x the programming time
- Easy full control of all sound, hi res, graphics, color, sprite, plotting line & circle
- Controllable SPLIT-SCREEN Display
- Includes interactive interpreter & compiler
- Forth virtual memory
- Full cursor Screen Editor
- Provision for application program distribution without licensing
- FORTH equivalent Kernel Routines
- Conditional Macro Assembler
- Meets all Forth 79 standards*
- Source screens provided
- Compatible with the book "Starting Forth" by Leo Brodie
- Access to all I/O ports RS232, IEEE, including memory & interrupts
- ROMABLE code generator
- MUSIC-EDITOR
- SPRITE-EDITOR
- Access all C-64 peripherals including 4040 drive
- Single disk drive backup utility
- Disk & Cassette based. Disk included
- Full disk usage—680 Sectors
- Supports all Commodore file types and Forth Virtual disk
- Access to 20K RAM underneath ROM areas
- Vectored kernel words
- TRACE facility
- DECOMPILER facility
- Full String Handling
- ASCII error messages
- FLOATING POINT MATH SIN/COS & SQRT
- Conversational user defined Commands
- Tutorial examples provided, in extensive manual
- INTERRUPT routines provide easy control of hardware timers, alarms and devices
- USER Support

SUPER FORTH 64™ is more powerful than most other computer languages!
* SUPERFORTH64 *



A SUPERIOR PRODUCT in every way! At a low price of only

\$96

Call: (415) 651-3160

PARSEC RESEARCH
Drawer 1776, Fremont, CA 94538

Take this ad to your local dealer, or B. Dalton Bookstore. Phone orders also accepted. Immediate delivery! Dealer inquiries invited. CA residents must include tax.

© PARSEC RESEARCH (Established 1979)

Commodore 64 & VIC-20 TM of Commodore



CIRCLE 49 ON READER SERVICE CARD

A comparison of Pascal compilers

By Namir Clement Shamas

Pascal's popularity has lead to the development of many commercial packages. All of the implementations have extensions to make one or more aspects of the language more versatile.

Which package is the best? What are the differences? Is the implementation you are using really appropriate for you, or did you buy it because of the brand name? If you're just learning Pascal, which is the one to select? Will the price affect your decision to purchase a particular version?

These questions lead me to carry out this review. I will compare six Pascal compilers for the IBM PC. The aspects of comparison will include the language implementation, compilation speed, code size and speed of program, and suitability for big software projects. The compilers I will review (by no means all the Pascal compilers on the market) are:

■ **MS-Pascal**, a product of Microsoft, was first released in 1981 for the IBM PC, using IBM labels. The compiler has gone through two major revisions. One was to accommodate the use of the 8087 numeric coprocessor chip. Microsoft is also directly selling its compiler for micros running MS-DOS, such as the IBM PC, Compaq and HP-150. The language implementation is rich with additional functions and procedures to perform a variety of low- and high-level data manipulations. The package is one of the "heavyweights."

■ **Pascal MT+** is sold by Digital Research. The language has an earlier implementation in the CP/M-80 environment and thus has been on the market for a few years for the Z80 machines. The compiler was more recently implemented for the CP/M-86, PC-DOS and CP/M68K. This implementation offers numerous functions and procedures to perform high- and low-level data manipulation. This is another extensive package.

■ **SBB Pascal** is a product of Software Building Blocks. The implementation stems from a previous 8-bit version sold as Pascal/Z. The latter has a good number

of public domain software and user groups.

■ **Turbo Pascal** is a product of Borland International. The product first appeared in 1983 and is marked by its low, affordable price. This fact and Turbo Pascal's speed of compilation and execution are the ingredients of its popularity and success. This product is excellent for "getting one's feet wet" with Pascal. It is very suitable for small projects.

■ **Practical Pascal** comes from the Canadian firm, Network Consulting, a licensee of Softech MicroSystem. It has been selling the UCSD Pascal and p-system for the IBM PC and compatibles. The company has made a number of smart improvements over the original UCSD Pascal (version IV) sold by Softech. Practical Pascal is an affordable, scaled-down version of the company's full package. This product is suitable for those who want or need to use software developed for the UCSD Pascal.

■ **Utah Pascal** is sold by Ellis Computing. The product is essentially a repackaged version of the famous JRT Pascal, a p-code implementation. The company has been selling for a few years its low-price Nevada software line for the CP/M environment. This line includes COBOL, FORTRAN, BASIC, PILOT and Pascal. Utah Pascal seems to be the first in a series of Utah software, similar to the Nevada series, aimed at the MS-DOS environment.

The original JRT Pascal first appeared for the CP/M environment. It was one of the early low-cost software packages. JRT

systems filed Chapter 11 but seems to be willing to sell its Pascal through Ellis Computing. The product is suitable for the novice who does not mind the slowness of the p-code execution.

Table 1 shows some basic data concerning the contents of the commercial packages. This information includes the compiler, linker, assembler, debugger, editor and whether the compiler produces native code or pseudocode.

All the compilers have implemented extensions to ISO Pascal. They vary from one package to another. Table 2 shows a comparison of implemented data types.

Integers. MS-Pascal and Pascal MT+ implement the most extensions on data types. This includes wider integer ranges used for higher addresses. Both have the *WORD* type with a range from 0 to 65,535. Both offer, together with Turbo Pascal, the absolute data type—variables declared to reside at specific memory locations. SBB Pascal has adopted the basic standard types. The predefined *ADDRESS* type is offered by MS-Pascal.

Others, as Table 2 shows, use a function call to return the address of a variable. This extension in integer types is aimed at making the implementations versatile for low-level data manipulations.

Reals. MS-Pascal, Pascal MT+, SBB Pascal and Practical Pascal offer either long reals, BCD reals or both. This is aimed at scientific, engineering and financial applications.

Basic data

	MS	MT+	SBB	Turbo	Utah	Practical
Compiler	yes	yes	yes	yes	yes	yes
Linker	yes	yes	no	no	yes	yes
Assembler	no	yes	no	no	no	yes
Debugger	no	yes	yes	no	yes	yes
Editor	no	no	yes	yes	no	yes
Version	3.2	3.2	3.0	2.0	4.1	4.1
Produce machine code	yes	yes	yes	yes	no	no

Table 1.

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

\$199⁰⁰

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

The support for the 8087 numeric coprocessor chip is unanimous for compilers generating native machine code. It allows floating point calculations to execute at a higher speed since all floating point arithmetic and function calculations are "hard-wired."

The benchmark tests presented later show, for example, that Pascal MT+ executes trigonometric functions 114 times faster with an 8087 chip than without it. Many implementations have recently added and/or improved the support for using the 8087.

Strings. MS-Pascal allows for a string variable to be up to 32KB long. This is second only to Utah Pascal which allows for 64KB strings. The rest maintain the standard maximum length of 256 bytes per string.

Open and conformant arrays. Pascal is a strongly-typed language, which has advantages and disadvantages. Disadvantages include limitations occurring when procedures and functions deal with arrays. The variables passed as arguments must be of the same type and size as stated in the routine declaration.

This poses a problem for programmers wishing to develop general purpose routines, such as matrix operations. To overcome the size limitations, dynamic arrays have been implemented in two ways.

The first, used by MS-Pascal, declares an upper array type with an unspecified upper bound. Variables can be declared as the upper array type with the upper limit specified. The same upper array type can be used in declaring the arguments of a function or procedure. MS-Pascal has predefined functions to obtain the array's limits.

Utah Pascal uses a similar technique: declaring dynamic array types. The variables can be declared with an unspecified bound too. The procedures *ALLOCATE* and *DEALLOCATE* are used to create and remove dynamic variables, respectively. These procedures can be used inside user functions and procedures.

The second method is to use conformant arrays. They are declarations appearing in the function or procedure arguments defining the type and dynamic size. Integer or character identifiers are used to specify the bounds of the arrays.

Data types

	MS	MT+	SBB	Turbo	Utah	Practical
Binary numbers	yes	no	yes	no	no	no
Octal numbers	yes	no	yes	no	no	no
Hex numbers	yes	yes	yes	yes	yes	no
WORD (0 to 65,535)	yes	yes	no	no	no	no
Byte (0 to 255)	yes	yes	no	yes	no	no
ADDRESS	yes	func	no	func	func	no
Absolute type	yes	yes	no	yes	no	no
Long integers	yes	yes	no	no	no	yes
Long reals	yes	no	no	no	no	yes
BCD reals	no	yes	yes	no	no	no
8087 support	yes	yes	yes	yes	no	no
Structured constant	yes	no	no	no	no	no
Sub-range type	yes	yes	yes	yes	yes	yes
Enumeration type	yes	yes	yes	yes	no	yes
Set type	yes	yes	yes	yes	yes	yes
Open arrays	yes	no	no	no	yes	no
Conformant arrays	no	yes	yes	yes ¹	no	no
Max string size (bytes)	32K	256	256	256	64K	256

1. Only for strings.

Table 2.

CIRCLE 16 ON READER SERVICE CARD

Pascal MT+ and SBB Pascal use this method. Multidimensional dynamic matrices are also implemented. Turbo Pascal has tackled the problem for dynamic strings only. It uses a compiler directive to relax the parameter type checking on strings. Thus strings of any length can be passed.

All the Pascal implementations have a similar program component. MS-Pascal introduces the VALUE section, where variables declared in the VAR section are assigned initial value.

All implementations except Practical Pascal introduce the ELSE or OTHERWISE clause to the CASE construct. MS-Pascal, Pascal MT+ and SBB Pascal

allow for functions to be passed as procedural parameters. This is valuable in writing procedures that process user-defined functions.

A simple example is developing procedures in numerical analysis to perform numerical integration or root seeking. Table 3 compares programming elements. Both Pascal MT+ and Turbo Pascal have procedures to handle in-line machine code and interrupts. These offer effective low-level machine access techniques.

Most of the implementations allow for DOS calls. SBB Pascal barely mentions this capability. MS-Pascal, Pascal MT+ and Turbo Pascal allow for bit and byte manipulation to benefit low-level data handling applications. Graphics are offered by SBB Pascal, Turbo Pascal and Practical Pascal. The last two offer cursor

Programming elements

	MS	MT +	SBB	Turbo	Utah	Practical
Procedural parameter	yes	yes	yes	no	no	no
ELSE clause in CASE statement	yes	yes	yes	yes	yes	no
In-line machine code	no	yes	no	yes	no	no
Interrupt procedure	no	yes	no	yes	no	no
MARK/RELEASE	yes	yes	yes	yes	yes	yes
NEW/DISPOSE	yes	yes	yes	yes	yes	yes
ROM call	no	no	yes	yes	no	no
DOS call	yes	yes	no	yes	yes	no
Bit/byte manipulation	yes	yes	no	yes	no	no
Byte/char manipulation	yes	yes	no	yes	no	no
String manipulation	yes	yes	yes	yes	yes	yes
Graphics	no	no	yes	yes	no	yes
Windows	no	no	no	yes	no	yes
Screen cursor control	no	no	no	yes	no	yes

Table 3.

I/O operations

	MS	MT +	SBB	Turbo	Utah	Practical
DOS 2.0 file support	yes	no	no	no	no	no
GET/PUT	yes	yes	yes	no	yes	yes
Port/I/O	no	yes	no	yes	yes	yes
Untyped file	no	yes	no	yes	no	yes

Table 4.

**ATTENTION:
ENGINEERS
PROGRAMMERS**

PolyFORTH® II

the powerful multitasking/
multi-user operating system
is now available for most
micro-computers running—

**CP/M-80
and
CP/M-86**

Offers CP/M users:

- An ability to run multiple terminals
- Unlimited control tasks
- Concurrent printer operation

These advanced features combine with FORTH, Inc.'s powerful version of the FORTH programming language to offer CP/M users the ideal environment for all interactive and real-time applications.

Featuring speed of operation, shortened development time, ease of implementation and overall cost-effective performance, this system is fully supported by FORTH, Inc.'s:

- Extensive on-line documentation
- Complete set of manuals
- Programming courses
- The FORTH, Inc. hot line
- Expert contract programming and consulting services

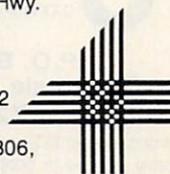
From FORTH, Inc., the inventors of FORTH, serving professional programmers for over a decade.

Also available for other popular mini and micro computers.

For more information contact:

FORTH, Inc.

2309 Pacific Coast Hwy.
Hermosa Beach,
CA 90254
213/372-8493
RCA TELEX: 275182
Eastern Sales Office
1300 N. 17th St. #1306,
Arlington, VA 22209
703/525-7778



*CP/M is a registered trademark of Digital Research

CIRCLE 30 ON READER SERVICE CARD

DeSmet C

8086/8088
Development
Package **\$109**

FULL DEVELOPMENT PACKAGE

- Full K&R C Compiler
- Assembler, Linker & Librarian
- Full-Screen Editor
- Execution Profiler
- Complete **STDIO** Library (>120 Func)

Automatic DOS 1.X/2.X SUPPORT

BOTH 8087 AND SOFTWARE FLOATING POINT

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 _____

CWARE
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.

CIRCLE 18 ON READER SERVICE CARD

NGS FORTH

*A FAST FORTH
OPTIMIZED FOR THE IBM
PERSONAL COMPUTER
AND MSDOS COMPATIBLES.*

- *79 STANDARD
- *FIG LOOKALIKE MODE
- *PC-DOS COMPATIBLE
- *ON-LINE CONFIGURABLE
- *ENVIRONMENT SAVE & LOAD
- *MULTI-SEGMENTED
- *EXTENDED ADDRESSING
- *AUTO LOAD SCREEN BOOT
- *LINE AND SCREEN EDITORS
- *DECOMPILER & DEBUGGING AIDS
- *8088 ASSEMBLER
- *BASIC GRAPHICS & SOUND
- *NGS ENHANCEMENTS
- *DETAILED MANUAL
- *INEXPENSIVE UPGRADES
- *NGS USER NEWSLETTER

*A COMPLETE FORTH
DEVELOPMENT SYSTEM.*

PRICE: \$70

*PLEASE INCLUDE \$2 POSTAGE &
HANDLING WITH EACH ORDER.*

*CALIFORNIA RESIDENTS :
INCLUDE 6.5% SALES TAX.*



NEXT GENERATION SYSTEMS
P.O. BOX 2987
SANTA CLARA, CA. 95055
(408) 241-5909

CIRCLE 45 ON READER SERVICE CARD

control in text mode and some window capability in the graphics mode. Turbo Pascal allows text windows too.

I/O operations. Table 4 is a comparison of I/O features. Only MS-Pascal has support for an MS-DOS 2.0 file structure. Its availability is expected since Microsoft authored MS-DOS.

For random I/O most implementations use *GET* and *PUT* to write and read records, respectively. The procedure *SEEK* is used to select the address of the target record. Turbo Pascal allows for the record number to be included in the *READ* and *WRITE* verbs.

Pascal MT+, Turbo Pascal, and Practical Pascal allow for untyped file I/O using *BLOCKREAD* and *BLOCKWRITE*. They allow for reading and writing a specific number of bytes. The same compilers allow for port I/O. This is useful in developing software for communications.

Perhaps using Pascal in large software projects is the acid test to show its versatility. This involves developing external general purpose functions, library modules, and units. Chaining programs and the use of overlays are other program segmentation methods. Table 5 compares segmentation capabilities.

All implementations allow for external functions. Pascal MT+ and Turbo Pascal allow for chaining where global or absolute variables can be passed from one program to another. Chaining with SBB Pascal is not as versatile. Data should be passed via data files.

Pascal MT+ implements powerful and flexible procedure overlays. Up to 255 overlays, up to 15 separate overlay areas, and nested overlays are supported. Nested overlays are also supported by Turbo Pascal but not by SBB Pascal.

MS-Pascal and Pascal MT+ allow for modules, a collection of procedures and functions without a main body.

MS-Pascal and Practical Pascal implement the UNIT libraries. They are composed of the *INTERFACE* and *IMPLEMENTATION* sections. Units are separately compiled modules and offer a powerful method for program segmentation and building routine libraries. Other programs call these units via the *USE* declaration.

Compiling with the six implementations ranges from very simple to a multistage process. A process requiring several steps reflects the compiling and linking options offered by the implementation.

Table 6 compares the number of stages to compile, whether a linker is invoked, and the optional use of an optimizer. At the linking stage, the user has a choice of linked libraries, such as the floating point support and transcendental functions libraries.

A software developer may ask the following practical question. How friendly is the implementation in handling compilation and run-time errors?

I have tested two types of code errors in some test programs: using an undeclared variable and adding an integer variable to a floating point constant. The run-time error that I tested is a division-by-zero error while attempting to divide two reals.

Here is a brief rundown of how each implementation reacts:

- MS-Pascal will display the code line containing the error and will point at the error with a message. For run-time errors, a message will be displayed and the program halts, leading you back to DOS.

- Pascal MT+ will display the error number and the line number on which it occurred. For run-time errors, a message is displayed inquiring whether you want to continue or abort.

- SBB Pascal displays a very brief error message. You have to list the LST file to see the location and error type. During my division-by-zero run-time test, the program just kept going! It did not bother to tell me that I had just committed a mathematical sacrilege.

- Turbo Pascal is the friendliest. For both error code and run-time errors, the system displays an error message and then invokes the editor to point out where things went wrong.

- Utah Pascal outputs the processed code lines to either the screen, printer, or file. It shows a message below the error-containing line. The message lacks a little bit of clarity. It does not point out the error. You have to examine the code line to discover where the error is. For the run-time error, the program displayed a warning about error in floating point operation, but the program kept going.

- Practical Pascal is similar to Turbo Pascal in reacting to an error found during compilation. Handling run-time error is done using very vague messages.

The review of the compilers included a benchmark test to compare the speed of compilation, size of executable code, and execution time. I carried the tests using an IBM XT with 512K bytes of memory and an 8087 chip.

All the compilers, except Practical Pascal, ran from the hard disk. Practical Pascal sets up a RAM drive upon booting. A single drive was used. The Hewlett-Packard HP41CV programmable calculator with a Timer Module was used for timing. The benchmark programs were:

- **The Sieve of Eratosthenes.** This is a very popular benchmark test used for a

Program segmentation

	MS	MT+	SBB	Turbo	Utah	Practical
Include	yes	yes	yes	yes	yes	yes
External functions	yes	yes	yes	yes	yes	yes
Chaining	no	yes	yes	yes	no	no
Overlays	no	yes	yes	yes	no	no
Modules	yes	yes	yes	no	no	no
Units	yes	no	no	no	no	yes

Table 5.

Compiling steps

	MS	MT+	SBB	Turbo	Utah	Practical
Compiler steps	2	1	2	1	1	1
Link	yes	yes	yes	no	opt.	opt.
Optional optimizer	yes	no	yes	no	no	yes

Table 6.

**ATTENTION:
ENGINEERS
PROGRAMMERS**

PolyFORTH® II

the operating system and programming language for real-time applications involving **ROBOTICS, INSTRUMENTATION, PROCESS CONTROL, GRAPHICS** and more, is now available for...

IBM PC*

PolyFORTH II offers IBM PC users:

- Unlimited control tasks
- Multi-user capability
- 8087 mathematics co-processor support
- Reduced application development time
- High speed interrupt handling

Now included at no extra cost: Extensive interactive **GRAPHICS SOFTWARE PACKAGE!** Reputed to be the fastest graphic package and the only one to run in a true multi-tasking environment, it offers point and line plotting, graphics shape primitives and interactive cursor control.

PolyFORTH II is fully supported by FORTH, Inc.'s:

- Extensive on-line documentation
- Complete set of manuals
- Programming courses
- The FORTH, Inc. hot line
- Expert contract programming and consulting services

From FORTH, Inc., the inventors of FORTH, serving professional programmers for over a decade.

Also available for other popular mini and micro computers.

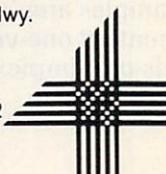
For more information contact:

FORTH, Inc.

2309 Pacific Coast Hwy.
Hermosa Beach,
CA 90254
213/372-8493
RCA TELEX: 275182

Eastern Sales Office
1300 N. 17th St.
Arlington, VA 22209
703/525-7778

*IBM PC is a registered trademark of International Business Machines Corp.

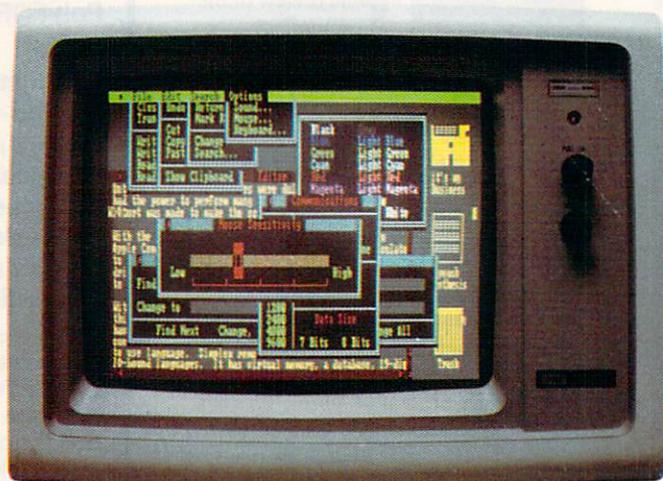


CIRCLE 37 ON READER SERVICE CARD

We thought about calling it MacSimplex . . . after all it makes your IBM® PC behave like a Macintosh™ and much more . . .

and with over two years in the making, the Simplex Database Management System has features like 32-megabyte virtual memory and the most powerful networked/relational database in the microcomputer industry. Simplex was designed around how you think and the Macintosh way, so that you can use your favorite mouse to handle those mundane tasks like menu selection and data manipulation. And, if you don't have a mouse, you can use our keyboard mouse simulator, MouSim™.

Pop-up and pull-down menus, dialog and alert boxes are not just added features, they are the heart of the Simplex way. In addition, Simplex gives you both a software and a hardware floating point capability, each with 19-digit accuracy. It permits login, password, privilege, and can be used on a local area network. Simplex has full communications and a remote or local printer spooler. Above all, Simplex is modular and grows with you! Simplex also has a full-featured, English-like language which is simple to use.



You can't buy Simplex™, but it is now available as an integral part of
it's my Business™ and will be used by *it's my Word™*, *it's my Graphics™*, . . .

Businessmen! *it's my Business* will revolutionize the way that you handle your business. It saves time, money, and standardizes your system for all who use it. *it's my Business* comes with applications like accounting, interoffice or intraoffice mail, editing, invoicing, inventory management, mail list, calendar, scheduler, forms and more. You can modify each of these to create applications specifically designed for you... maybe we should have called it "it's your Business".

Professionals! *it's my Business* has over 200 pages of examples and demonstrations to show you how to solve your everyday professional problems. And if these examples aren't enough, we give you a complimentary one-year subscription to Questalk™, our hands-on Simplex applications magazine.

System integrators and consultants, beware! If you are not using *it's my Business* with Simplex to solve your problems, don't be surprised when more novice programmers solve that complex math, industrial engineering, or business problem faster. We think that you can cut your concept-to-development time by an order of magnitude!

it's my Business (includes *it's my Editor*) - \$695.00
it's my Business Demo Disk - \$20.00
it's my Editor - \$100.00.

Quest Research software is available through your local computer store or through mail order from Quest Software Corporation at (205) 539-8086, 303 Williams Avenue, Huntsville, AL 35801.

Value added resellers and dealers please contact Quest Research, Incorporated at (800) 558-8088, 303 Williams Avenue, Huntsville, AL, 35801.

Quest™
Quest Research Inc.

IBM is a registered trademark of International Business Machines. Macintosh is a trademark of Apple Corporation. *it's my Business*, *it's my Word*, *it's my Graphics*, *it's my Editor*, *it's my Home*, *it's my Voice*, *it's my Ear*, *it's my Statistics*, Simplex, MouSim, Questalk, and the Quest logo are trademarks of Quest Research, Incorporated.

variety of computers and languages.

■ **Integer array sorting.** This program creates an orderly array of 1,000 integers. Then the array is sorted to reverse the initial order. The time for the sorting process is measured.

■ **Matrix inversion.** This is the first test program to examine the speed of floating point arithmetic. The program creates a 20 by 20 matrix with ones in the non-diagonal elements and twos in the diagonal elements. The time needed to invert the matrix is measured. The program is recompiled to use the 8087 support when possible.

■ **Trigonometric test.** A program for comparing the computing speed of the sine and cosine function 1,000 times each.

Again, the program is recompiled to use the 8087 support when possible.

Table 7 shows the results for each test program. The compiling time recorded includes the time for the number of compiler passes plus the linking time.

The timer is stopped at the end of each stage and resumed as the next one starts in a multistage situation. It is worthwhile to point out that MS-Pascal automatically detects the presence of the 8087 chip and uses the 87 code library. All other compilers supporting the 8087 do so at the users command.

The results of the sieve test program show the following:

■ The compactness of the Utah Pascal p-code

Benchmark test results Sieve test

	Source/compiled code size (bytes)	Compiling and linking time (min:sec)	Execution time (min:sec)
MS	659/27,708	1:02	0:11.5
MT+	768/10,752	0:41	0:15.6
SBB	659/5,120	0:33	0:15.0
Turbo	659/10,299	0:03	0:15.0
Utah	659/384	0:25.4	8:15.0
Practical	4 blocks/2 blocks	0:17	2:26.0

Integer sort test

	Source/compiled code size (bytes)	Compiling (and linking) time (min:sec)	Execution time (min:sec)
MS	1,119/21,692	1:05	0:01.5
MT+	1,152/10,752	0:42	0:03
SBB	1,119/5,248	0:35	0:02
Turbo	1,119/10,454	0:03	0:02
Utah	1,119/512	0:29	1:05
Practical	4 blocks/2 blocks	0:18	0:21

Matrix inversion test

	Source/compiled code size (bytes)	Compiling and linking time (min:sec)	Execution time (min:sec)
MS	1,792/34,984	1:49	0:2.84
MT+			
without 87	1,792/18,944	0:56	2:10
with 87	116,384	0:51	0:06.4
SBB			
without 87	1,792/10,112	0:42	0:43
with 87	8,320	0:43	0:02.2
Turbo			
without 87	2,793/11,108	0:03.5	error
with 87	10,027	0:03.5	0:05.39
Utah	1,792/896	0:36	2:12
Practical	4 blocks/3 blocks	0:24	0:20

Table 7. (Continued on a following page).

**ATTENTION:
ENGINEERS
PROGRAMMERS**

PolyFORTH® II

the operating system and programming language for real-time applications involving **ROBOTICS, INSTRUMENTATION, PROCESS CONTROL, GRAPHICS** and more, is now available for...

DEC* PDP-II* and LSI-II* Systems

The PolyFORTH II high performance features include:

- Multiple users (30 terminals on a LSI-II)
- Unlimited control tasks
- High speed interrupt handling
- Reduced application development time

PolyFORTH II software will run on any standard PDP* or LSI-II with RX02 disk (RSX* optional), Micro/PDP-II* and PROFESSIONAL* 350 and is fully supported by FORTH, Inc.'s:

- Extensive on-line documentation
- Complete set of manuals
- Programming courses
- The FORTH, Inc. hot line
- Expert contract programming and consulting services

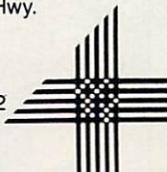
From FORTH, Inc., the inventors of FORTH, serving professional programmers for over a decade.

Also available for other popular mini and micro computers.

For more information contact:

FORTH, Inc.

2309 Pacific Coast Hwy.
Hermosa Beach,
CA 90254
213/372-8493
RCA TELEX: 275182
Eastern Sales Office
1300 N. 17th St.
Arlington, VA 22209
703/525-7778



*Registered trademarks of Digital Equipment Corp.

CIRCLE 40 ON READER SERVICE CARD

- The size of the code from Pascal MT+ and Turbo Pascal is almost identical and so is the execution time
- SBB Pascal has produced the smallest non-p-code file sizes
- The compilation speed of Turbo Pascal
- Small variation in execution time for non-p-code programs— MS-Pascal is slightly faster.
- The Practical Pascal program is faster than that of Utah-Pascal.

The conclusions drawn from the results of the integer sorting test seem to agree with those of the sieve test.

With the matrix inversion, the Turbo Pascal compiler version that does not support the 8087 gave a run-time error. All other compilers, including the 8087 Turbo Pascal version, produced executable code that ran smoothly. The conclusions to be drawn are:

Not using the 8087

- Pascal MT+ is almost as fast as Utah Pascal. This indicates the floating point operation of Pascal MT+ may be written in the same language.
- Practical Pascal has faster math operation than SBB Pascal. The reason may be similar to the previous one.

Sine and cosine trigonometric test

	Source/compiled code size (bytes)	Compiling (and linking) time (min:sec)	Execution time (min:sec)
MS	902/20,934	1:21	(sin) 0:10.79 (cos) 0:10.80
MT+	1,024/22,528	0:58	(sin) 22:29 (cos) 32:46
	16,896	0:51	(sin) 00:11.8 (cos) 00:11.8
SBB	902/9,856	0:39	(sin) 5:38 (cos) 5:49
	1,024/8,064	0:38	(sin) 0:09.6 (cos) 0:11.5
Turbo	902/10,620	0:02.89	(sin) 5:01.5 (cos) 5:06.4
	9,565	0:02.86	(sin) 0:10.56 (cos) 0:11.05
Utah	1,024/640	0:29.5	(sin) 10:08.67 (cos) 10:53
Practical	4 blocks/3 blocks	0:39	(sin) 1:52.36 (cos) 1:57.88

Note: A block of storage for the UCSD p-system has 512 bytes.

Table 7. (Continued from a preceding page).

UNIX
TIMESHARE+

*UNIX System III POWER and sophistication are yours. Let THE SOLUTION turn your micro into all you dreamed it could be, bringing the Ultimate programming environment as close as your modem. Just a local call from over 300 cities nationwide via Telenet.

THE SOLUTION™

- **EXPANSIVE SOFTWARE DEVELOPMENT FACILITIES** including Language and Operating System design.
- **LANGUAGES:** C, Fortran 77, RATFOR, COBOL, SNOBOL, BS, Assembler + Artificial Intelligence programming via LISP.
- **USENET Bulletin Board System**—800+ international UNIX sites feeding over 190 categories, typically bringing you more than 160 new articles per day.
- **Interuser and Intersystem mail** + 'chat' capability.
- **UNIFY:** Sophisticated data-base management system.
- **UNIX & System enhancements** from U.C. Berkeley and Korskmeier Electronic Design Inc.
- **Online UNIX manuals** + Expert consultation available.
- **SOLUTION-MART:** Hardware/Software discount shopping database.
- **LOW COST and FAST** response time.
(as low as \$8.95 hr. connect time + \$.05 cpu sec. non-prime)
- **\$24.95 = 1 hr. FREE system time** + SOLUTION News subscription + **BYTE BOOK** (Introducing The UNIX System 556 pp.).

*UNIX is a trademark of Bell Labs.



Payment via VISA or MasterCard

Korskmeier
ELECTRONIC DESIGN, INC.
CIRCLE 34 ON READER SERVICE CARD

5701 Prescott Avenue
Lincoln, NE 68506-5155
402/483-2238
10a-7p Central

Only \$95 with FULL SOURCE CODE!



"... 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!



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

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, Epon, Morrow, Lobo, Zenith, Xerox. Please request our new catalog. We welcome COD orders.

Workman & Associates

112 Marion Avenue
Pasadena, CA 91106
(818) 796-4401



CIRCLE 68 ON READER SERVICE CARD

PROGRAMMER'S DEVELOPMENT TOOLS

IBM Personal Computer Language and Utility Specialists

LANGUAGES:

	List	Ours
8088 Assembler w/Z-80 Translator 2500 AD	100	89
C-86 by Computer Innovations	395	309
CB-86 by DRI	600	429
DeSmet C Compiler with Debugger	159	145
Instant-C by Rational Systems, <i>Interpretive C</i>	500	469
Janus/ADA+ Tools by R&R	700	499
Lattice C Compiler	500	299
STSC APL*Plus/PC	595	499
Xenix Development System by SCO	1350	1099

Call for Prices and Information about other Languages.

Special Holiday Season Sale Price!

Mark Williams C Development System \$429

The MWC Development System defines a new standard for C Compilers. The compiler provides excellent benchmarks and extremely fast compilations. In addition, the system includes its own assembler and source level debugger. Best of all, **you can now save an additional \$30 off** our already discounted price. Call for details.

Manufacturer's List Price \$500.

Special Introductory Offer

Save \$50 off manufacturer's list price!
BetterBASIC by Summit Software \$149

This new approach to BASIC programming supports modular structured techniques, large workspaces, incremental compilation, pointers and more. An optional 8087 Math Module is available for \$89.

UTILITIES:

	List	Ours
Btrieve by SoftCraft	245	199
CodeSmith-86 by Visual Age	145	129
Communications Library by Greenleaf	160	139
C-Food Smorgasbord	150	109
C Power Paks from Software Horizons	CALL	CALL
C-Tree by Faircom	395	359
C To Dbase by Computer Innovations	150	139
C-Utilities by Essential Software	Sale Priced!	149 119
Dr. Halo	100	89
ESP for C or Pascal by Bellesoft	CALL	CALL
FirsTime for C by Spruce Tech	295	269
GraphiC from Scientific Endeavors	195	179
Greenleaf Functions Library	Sale Priced!	175 139
Halo Color Graphics for Lattice, CI-86	200	125
Panel Screen Design/Editing by Roundhill	295	234
Pfix 86 by Phoenix Software	195	175
Pfix 86+ by Phoenix Software	395	355
Phact by Phact Associates	395	359
Plink-86 Overlay Linkage Editor	395	310
Pmate by Phoenix Software	225	175
Profiler by DWB & Associates	New Low Price!	125 99
Tools by Blaise Computing	CALL	CALL
Trace-86 by Morgan Computing	125	115
Windows for C by Creative Solutions	195	159

Prices are subject to change without notice.

Call for our New Catalog consisting of 200+ Programmer's Development Tools Exclusively for IBM PC's and Compatibles.

Account is charged when order is shipped.



1-800-336-1166



Programmer's Connection
281 Martine! Drive
Kent, Ohio 44240
(216) 678-4301 (In Ohio)

"Programmers Serving Programmers"

CIRCLE 54 ON READER SERVICE CARD

Elegance
Power
Speed



Users' Group
Supporting All C Users
Box 287
Yates Center, KS 66783

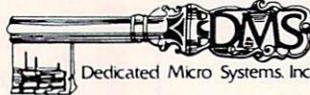
CIRCLE 10 ON READER SERVICE CARD

BDS C v1.5 For FAST Development

- Fastest compile to execute cycle of any CP/M 80® Compiler.
- Dynamic debugger.
- 180 page manual.

KS res. add 4% tax.
\$120.00
Plus \$2.50
Shipping & handling

TERMS: check, c.o.d., charge card.
(316) 431-0018



P.O. Box 481 Chanute, KS 66720

CP/M © trademark of Digital Research

CIRCLE 21 ON READER SERVICE CARD

New Release

CP/M ↔ **ISIS**
for
PDS & MDS

ICX v.4 eXchanger now supports BOTH 8" MDS and 5-1/4" iPDS formats. Manipulation of ISIS-II files using your CP/M system was never easier.

ISE v.6 Emulator gives the CP/M-80 user access to all the ISIS-II languages and utilities.

Complete source (C and MAC asm) included with all packages

ICXMSD \$89
ICXPDS \$89
ISE \$89
ICX Toolkit (all 3) \$250

Copyrights: CP/M Digital Research, Inc., ISIS II and iPDS Intel Corp.

Western Wares

303-327-4898 • Box C • Norwood, CO 81423

CIRCLE 75 ON READER SERVICE CARD

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

Have you tried **SNOBOL4+?**

MUSIC ANALYSIS • TEXT PROCESSING • SYMBOLIC MATHEMATICS • ARTIFICIAL INTELLIGENCE
COMPILE 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 5 1/4" DSDD, specify DOS/CPM format

Send check, VISA, M/C to: **\$95**

Catspaw, Inc. plus '3 s/h
P.O. Box 1123 • Salida, CO 81201 • 303/539-3884

CIRCLE 9 ON READER SERVICE CARD

*** EASY TO USE ***

Macro Programs for

Spellbinder

NOW for PC-DOS

Customized Tables -- RAM disk with .BAT file. Toggle extended character set and printer control.

Our old standby, DearJohn, mailing list management with utilities. Still \$67.50 CP/M or PC-DOS

"NEW" "All You Wanted to know about Spellbinder but Couldn't Find Out". A programmer's notebook of major commands and how to use. \$10.00 per copy

We have a long list of custom macros for use in the office and by the programmer. Send \$1.00 for catalog.

COMPUTER RESOURCES of WAIMEA
P.O. Box 1569 Kamuela, Hawaii 96743
(808) 885-7905

CIRCLE 14 ON READER SERVICE CARD

Interested in writing,
reviewing software,
or refereeing
manuscripts for

**COMPUTER
LANGUAGE**

For information contact:
Craig LaGrow/Editor
131 Townsend St.
San Francisco, CA 94107

(415) 957-9353
BBS#: (415) 957-9370
CompuServe Acct: GO CLM

Premier _____ copies × \$4.00 = \$ _____
Oct. '84 _____ copies × \$4.00 = \$ _____
Nov. '84 _____ copies × \$4.00 = \$ _____
Dec. '84 _____ copies × \$4.00 = \$ _____
Total \$ _____

NAME _____
COMPANY _____
ADDRESS _____
CITY, STATE, ZIP _____

Send payment and coupon to:
COMPUTER LANGUAGE
Back Issues
131 Townsend St.
San Francisco, CA 94107

ORDER BACK ISSUES OF COMPUTER LANGUAGE WHILE THEY LAST!

To receive your back issues, just fill out this coupon and mail it back with \$4.00 per issue.

The 8087 supported test

- SBB Pascal has the fastest executing code (sorry Turbo), with MS-Pascal a close very second.
- Using the 8087 for math operation with Pascal MT+ is a must.

The trigonometric test helped us to draw the following conclusions.

Not using the 8087

- Turbo Pascal has the fastest executing code, followed by SBB Pascal, a close second.
- Pascal MT+ performed the poorest in trig functions. Even Utah Pascal is twice as fast.

The 8087 supported test

- A very slight variation was evident in the speed of running code. SBB Pascal has the fastest execution of a sine function. Turbo Pascal has the lead for cosine functions. Basically all 8087 generated code runs at about the same speed.

At the end of this review, I can say that the six implementations examined offer something for everyone.

■ The novice Pascal programmer who does not want to spend a lot of money will enjoy Utah Pascal or Turbo Pascal. Turbo Pascal will probably be more attractive due to its speed, built-in editor, graphics and popularity.

■ For a programmer interested in using a UCSD implementation, Practical Pascal would be an excellent choice. It will allow a good degree of compatibility with, say, Apple UCSD Pascal.

■ The professional programmer working on large software projects and/or those involving low level machine access would make use of MS-Pascal or Pascal MT+.

■ The professional programmer working on scientific, engineering, statistical and financial applications would find MS-Pascal, Pascal MT+ and SBB Pascal suitable. MS-Pascal has versatile utilities written by other software companies.

■ Turbo Pascal, characterized by its speed of compilation and execution, can be used for front-end software development and testing. Batches of routines are developed and tested. Once the code is working properly, it is incorporated in a bigger implementation. **■**

Pascal and C Programmers

Your programs can now compile the

FirstTime™

FirstTime 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. *FirstTime* then positions the cursor over the error so you can correct it easily. *FirstTime* will identify all syntax errors, undefined variables, and even statements with mismatched variable types. In fact, any program developed with the *FirstTime* editor will compile on the first try.

Unprecedented

FirstTime 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 *FirstTime*. 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.

<i>FirstTime</i> for PASCAL	\$245
<i>FirstTime</i> 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 *FirstTime*. (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.



CIRCLE 33 ON READER SERVICE CARD

Information on the six Pascal compilers

Product name	Available from	Price	Hardware required
MS-Pascal	Microsoft Corp., 10700 Northup Way, P.O. Box 97200, Bellevue, Wash. 98009, (800) 426-9400	\$300.00	IBM PC and compatibles, and all other MS-DOS machines.
Pascal MT+	Digital Research Inc., P.O. Box DRI, Monterey, Calif. 93942, (408) 649-3896	\$400.00	Any 8086- or 8088-based machine running either PC-DOS or CP/M-86. Minimum 192K; 256K recommended.
SBB Pascal	Software Building Blocks Inc., P.O. Box 119, Ithaca, N.Y. 14851, (607) 272-2807	\$350.00	IBM PC and compatibles, and Z8000-based machines.
Turbo Pascal	Borland International, 4113 Scotts Valley Dr., Scotts Valley, Calif. 95066, (800) 227-2400 ext. 968	\$49.95	Any Z80- or 8086-based machine running either MS-DOS, PC-DOS, CP/M-86 or CP/M-80.
Practical Pascal	Network Consulting Inc., P.O. Box 8040, Blaine, Wash. 98230, (604) 430-3466	\$145.00	IBM PC, PCjr, or most compatibles.
Utah Pascal	Ellis Computing Inc., 3917 Noriega St., San Francisco, Calif. 94122, (415) 753-0186	\$39.95	IBM PC, XT, or AT with minimum 128K.

ADVERTISER INDEX

	PAGE NO.	CIRCLE NO.		PAGE NO.	CIRCLE NO.
ACM	19	7	Mendocino Software Co., Inc.	13	48
Alcor Systems	64	1	Micro Methods	22	52
Alpha Computer Service	64	2	MicroTec Research	8	77
Atron	4	4	MicroWay	20	57
Awareco	67	3	Microcompatibles	20	15
BD Software	10	5	Miller Microcomputer Service	11	61
Borland International	6,7	6	Mindbank, Inc.	66	63
C Systems	70	16	Next Generation Systems	72	45
C Users Group	78	10	Northwest Computer Algorithms	31	46
C Ware	72	18	Opt-Tech Data Processing	31	66
Carousel Micro Tools	46	8	Opt-Tech Data Processing	40	67
Catspaw, Inc.	78	9	Parsec Research, Inc.	68	49
Code Works, The	77	42	Plum Hall	25	50
CompuPro	Cover IV	12	Poor Person Software	61	51
Computer Innovations	57	11	ProCode	67	53
Computer Resources of Waimea	78	14	Programmer's Connection	77	54
DWB Associates	17	20	Programmer's Shop	25	69
Datalight	68	19	QCAD	68	23
David Data	30	17	Quest Research Inc.	74	55
Dedicated Micro Systems, Inc.	78	21	RR Software	Cover II	58
Earth Computer	12	24	Rational Systems, Inc.	24	56
Ecosoft	11	22	Ream, Edward	12	27
Essential Software Inc.	23	28	SLR Systems	32	59
FairCom	17	29	Simpliway Products, Co.	13	70
Forth, Inc.	71	30	Software Horizons	62	25
Forth, Inc.	73	37	Software Toolworks	80	26
Forth, Inc.	75	40	Solution Systems	22	60
Goodyear Aerospace	1	43	Solution Systems	67	71
Greenleaf Software	16	44	Spruce Technology Corp.	79	33
HSC, Inc.	56	31	Stride Micro	Cover III	72
Harvard Softworks	65	47	Summit Software	13	62
Introl Corp.	60	32	Syntax Constructs Inc.	40	73
Korsmeyer Electronics Design Inc.	76	34	Systems Engineering Tools	68	64
Laboratory Microsystems, Inc.	11	35	Systems Guild	23	76
Lattice Inc.	61	36	TeleSoft	2	74
MBP Software & Systems Technology	18	39	Thunder Software	56	65
Megamax, Inc.	64	13	UniPress	32	38
			Western Ware	78	75
			Wordtech Systems	14	41
			Workman & Associates	77	68

The index on this page is provided as a service to our readers. The publisher does not assume any liability for errors or omissions.

"C/80 . . . the best software buy in America!"

—MICROSYSTEMS

Other technically respected publications like *Byte* and *Dr. Dobb's* have similar praise for **The Software Toolworks' \$49.95 full featured 'C' compiler for CP/M® and HDOS with:**

- I/O redirection
- command line expansion
- execution trace and profile
- initializers
- Macro-80 compatibility
- ROMable code
- and much more!

"We bought and evaluated over \$1500 worth of 'C' compilers. . . C/80 is the one we use."

— Dr. Bruce E. Wampler
Aspen Software
author of "Grammatik"

In reviews published worldwide the amazing **\$49.95 C/80 from The Software Toolworks** has consistently scored at or near the top — even when compared with compilers costing ten times as much!

The optional **C/80 MATHPAK** adds 32-bit floats and longs to the C/80 3.0 compiler. Includes I/O and transcendental function library all for only **\$29.95!**

C/80 is only one of 41 great programs each **under sixty bucks**. Includes: LISP, Ratfor, assemblers and over 30 other CP/M® and MSDOS programs.

For your **free** catalog contact:

The Software Toolworks®
15233 Ventura Blvd., Suite 1118,
Sherman Oaks, CA 91403 or call 818/986-4885 today!

CP/M is a registered trademark of Digital Research.

CIRCLE 26 ON READER SERVICE CARD



NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

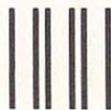
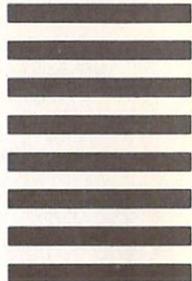
BUSINESS REPLY CARD

FIRST CLASS PERMIT NO. 27346 PHILADELPHIA, PA USA

POSTAGE WILL BE PAID BY

**COMPUTER
LANGUAGE**

P.O. BOX 11747
PHILADELPHIA, PA 19101



NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

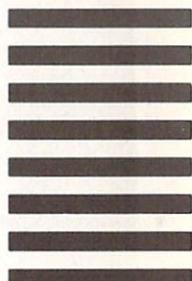
BUSINESS REPLY CARD

FIRST CLASS PERMIT NO. 27346 PHILADELPHIA, PA USA

POSTAGE WILL BE PAID BY

**COMPUTER
LANGUAGE**

P.O. BOX 11747
PHILADELPHIA, PA 19101



**COMPUTER
LANGUAGE**

Subscribe to **COMPUTER LANGUAGE** today for only \$24.00 — over 33% savings off the single copy price.

- Yes, start my Subscription to **COMPUTER LANGUAGE** today. The cost is only \$24.00 for 1 year (12 issues).
- I want to increase my savings even more — send me 2 years (24 issues) of **COMPUTER LANGUAGE** for only \$39.00.
- Payment enclosed Bill me

Name _____

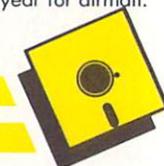
Company _____

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 \$30.00 per year. Outside the U.S., \$36.00/year for surface mail or \$54.00/year for airmail.

BID4



**COMPUTER
LANGUAGE**

Subscribe to **COMPUTER LANGUAGE** today for only \$24.00 — over 33% savings off the single copy price.

- Yes, start my Subscription to **COMPUTER LANGUAGE** today. The cost is only \$24.00 for 1 year (12 issues).
- I want to increase my savings even more — send me 2 years (24 issues) of **COMPUTER LANGUAGE** for only \$39.00.
- Payment enclosed Bill me

Name _____

Company _____

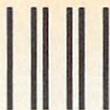
Address _____

City, State, Zip _____

Please allow 6-8 weeks for delivery of first issue. Foreign orders must be prepaid in U.S. funds. Canada orders \$30.00 per year. Outside the U.S., \$36.00/year for surface mail or \$54.00/year for airmail.

BID4





NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

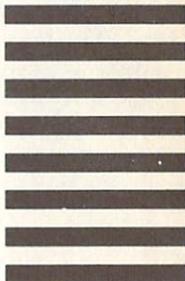
BUSINESS REPLY CARD

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



READER SERVICE CARD



Name _____
Company _____
Address _____
City, State, Zip _____
Country _____ Telephone number _____

Please complete these short questions:

1. I obtained this issue through:
- Subscription Passed on by associate
 Computer Store Other _____
 Retail outlet
2. Job Title _____
3. The 5 languages that I am most interested in reading about (list in order of importance).

December issue. Not good if mailed after April 30, 1985.

Circle numbers for which you desire information.

1	11	21	31	41	51	61	71	81	91
2	12	22	32	42	52	62	72	82	92
3	13	23	33	43	53	63	73	83	93
4	14	24	34	44	54	64	74	84	94
5	15	25	35	45	55	65	75	85	95
6	16	26	36	46	56	66	76	86	96
7	17	27	37	47	57	67	77	87	97
8	18	28	38	48	58	68	78	88	98
9	19	29	39	49	59	69	79	89	99
10	20	30	40	50	60	70	80	90	100

Comments _____

 Attn: Reader Service Dept. 1/4



NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

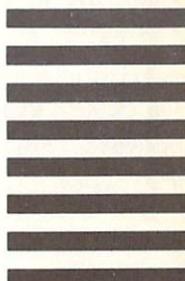
BUSINESS REPLY CARD

FIRST CLASS PERMIT NO. 22481 SAN FRANCISCO, CA USA

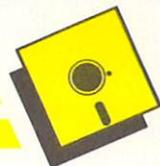
POSTAGE WILL BE PAID BY

COMPUTER LANGUAGE

2443 FILLMORE STREET • SUITE 346
SAN FRANCISCO, CA 94115



READER SERVICE CARD



Name _____
Company _____
Address _____
City, State, Zip _____
Country _____ Telephone number _____

Please complete these short questions:

1. I obtained this issue through:
- Subscription Passed on by associate
 Computer Store Other _____
 Retail outlet
2. Job Title _____
3. The 5 languages that I am most interested in reading about (list in order of importance).

December issue. Not good if mailed after April 30, 1985.

Circle numbers for which you desire information.

1	11	21	31	41	51	61	71	81	91
2	12	22	32	42	52	62	72	82	92
3	13	23	33	43	53	63	73	83	93
4	14	24	34	44	54	64	74	84	94
5	15	25	35	45	55	65	75	85	95
6	16	26	36	46	56	66	76	86	96
7	17	27	37	47	57	67	77	87	97
8	18	28	38	48	58	68	78	88	98
9	19	29	39	49	59	69	79	89	99
10	20	30	40	50	60	70	80	90	100

Comments _____

 Attn: Reader Service Dept. 1/4

"Despite the recent press notices, multiuser microcomputers aren't anything new!"

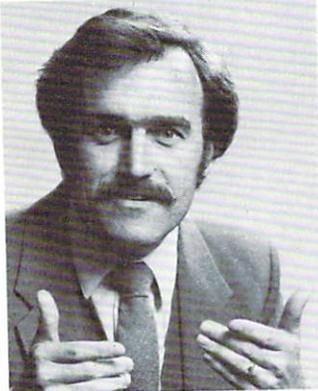
This is the first in a series of discussions with Rod Coleman, President of Stride Micro (formerly Sage Computer) on the 68000 multiuser market and its current environment.

Q: Why do you say that?

RC: "The technology to build a high performance multiuser system has been around for five years. And while some of the leaders in this industry have been pretending that micro multiuser didn't exist, we've been shipping complete systems for nearly three years. The benefits of multiuser are undeniable; it is more cost effective, and offers greater flexibility and utility. But until just recently, the marketing pressure to be compatible instead of being better, has blinded the industry."

Q: What do you mean?

RC: "Well, for example, the Motorola 68000 processor introduced 16/32-bit technology to the personal computer world a long time ago. It was fully capable of



"A surprising feature is compatibility. Everybody talks about it, but nobody does anything about it."

meeting high performance and multiuser design requirements in 1980. Instead of this trend taking off, most energy was spent promoting 8088/8086 products that

were clearly inferior from a technical point of view. This phenomenon leads me to believe that they will soon rewrite the old proverb: 'Build a better mousetrap and the world will beat a path to your door,' but only if they can find the way through the marketing fog."

Q: Are things changing now?

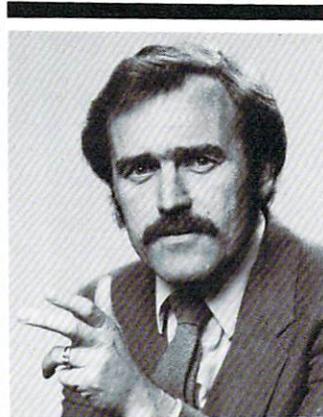
RC: "Yes and no. With the business world starting to take more and more interest in microcomputer solutions, the advantages of a solid multiuser system couldn't be kept hidden forever; companies like ours and a few others were beginning to make a dent. Instead of taking a fresh approach, some of the newest multiuser offerings will probably only give the technology an undeserved black eye! Multiuser is far more than the ability to plug in more terminals. It involves things like machine compatibility, fast processors, adequate memory, large storage capacities, backup features, networking, and operating system flexibility."

Q: Is this what makes the new Stride 400 Series different?

RC: "Exactly. That sounds self-serving, but it's true. Today a number of companies are introducing their first multiuser system. We've been building and shipping multiuser machines for almost three years. We know the pitfalls, we've fallen into some of them. But we have learned from our mistakes."

Q: Give me some examples.

RC: A hard disk is almost mandatory for any large multiuser installation. Yet, backing up a hard disk can be a nightmare if you only have floppies to work with. That's why we've added a tape backup option to all the larger Stride 400 Series machines. It's irresponsible for a manufacturer to market a multiuser system without such backup. Another good lesson was bus design. We started with one of our own designs, but learned that it's important not only to find a bus that is powerful, but also one that has good support and a strong future to serve tomorrow's needs. We



"The marketing pressure to be compatible instead of being better, has blinded the industry."

think the VMEbus is the only design that meets both criteria and thus have made it a standard feature of every Stride 400 Series machine."

Q: What are some of the other unique features of the 400 Series?

RC: "A surprising feature is compatibility. Everybody talks about it, but nobody does anything about it. Our systems are completely compatible with each other from the 420 model starting at \$2900, through the 440, on to the powerful 460 which tops out near \$60,000. Each system can talk to the others via the standard built-in local area network. Go ahead and compare this with others in the industry. You'll find their little machines don't talk to their big ones, or that the networking and multiuser are incompatible, or that they have different processors or operating systems, and so on."

Q: When you were still known as Sage Computer, you had a reputation for performance, is that still the case with the new Stride 400 Series?

RC: "Certainly, that's our calling card: 'Performance By Design.' Our new systems are actually faster; our standard processor is a 10 MHz 68000 running with no wait

states. That gives us a 25% increase over the Sage models. And, we have a 12 MHz processor as an option. Let me add that speed isn't the only way to judge performance. I think it is also measured in our flexibility. We support a dozen different operating systems, not just one. And our systems service a wide variety of applications from the garage software developer to the corporate consumer running high volume business applications."

Q: Isn't that the same thing all manufacturers say in their ads?

RC: "Sure it is. But to use another over used-term, 'shop around'. We like to think of our systems as 'full service 68000 supermicrocomputers.' Take a look at everyone else's literature and then compare. When you examine cost, performance, flexibility, and utility, we don't think there's anyone else in the race. Maybe that's why we've shipped and installed more multiuser 68000 systems than anyone else."



STRIDE
MICRO

Formerly Sage Computer

For more information on Stride or the location of the nearest Stride Dealer call or write us today. We'll also send you a free copy of our 32 page product catalog.

Corporate Offices:
4905 Energy Way
Reno, NV 89502
(702) 322-6868

Regional Offices:
Boston: (617) 229-6868
Dallas: (214) 392-7070

CIRCLE 72 ON READER SERVICE CARD

HERE TODAY HERE TOMORROW

When buying a computer, you can't limit yourself to just satisfying today's needs. **The best value in a system comes from its productivity . . . both for today and tomorrow.** CompuPro's System 816™ computer has that value. With all the power and capacity to handle your needs now and down the road.

System 816's longevity stems from top quality components . . . high storage capacity . . . the flexibility to handle a large variety of applications . . . and the speed to get the job done fast. Upgrading is easy, and when it's time to expand from single to multi-user operation, it's as simple as plugging in boards and adding terminals. Your system grows as you grow.

CompuPro also provides a library of the most popular software programs with your system and because it's CP/M® based, you have more than 3,000 other programs to choose from.

Even our warranty is for today and tomorrow. It spans 365 days — and includes the additional security of Xerox Americare™ on-site service nationwide for designated systems.*

What's more, CompuPro is one company you can count on to be around tomorrow. For more than ten years we've been setting industry standards, increasing productivity and solving problems.

For a free copy of our business computer buyer's primer, and the location of the Full Service CompuPro System Center nearest you, call (415) 786-0909 ext. 206.

CompuPro's System 816. The computer that's just as essential tomorrow as it is today.

CompuPro®

A GODBOUT COMPANY

3506 Breakwater Court, Hayward, CA 94545

* Available from Full Service CompuPro System Centers and participating retailers only.

System 816 and The Essential Computer are trademarks of CompuPro. CP/M is a registered trademark of Digital Research Inc. Americare is a trademark of Xerox Corporation.

System 816 front panel design shown is available from Full Service CompuPro System Centers only. © 1984 CompuPro

The Essential Computer™

CIRCLE 12 ON READER SERVICE CARD

