US NEWS AND REPORT

Issue Number
13
MARCH, 1985

Is your software easy to use?

Maybe YOU think so.

Despite what you may think, there may be times when your software is hard to use. **But what if your software could tell users exactly what to do every time they were confused?** Then people would start to agree with you about "easy to use".

And when everyone agrees with you, prospective customers become eager buyers, first-time users turn into confident users, reviewers give you high marks, and sales people enjoy demonstrating your software. **Quite simply, when everyone agrees your software is easy to use, it sells.**

SoftDoc™ is a module which provides your software with instantaneous context-sensitive help, on-line reference facilities and interactive tutorials. SoftDoc™ is compatible with windowing, networking, touch screens, mouse control and other interface technologies.

Call or write Learning Tools today. Find out how your software can become easier to use, demonstrate, learn, document, maintain, distribute, network, support, review and sell. And ask for a SoftDoc™ demonstration disk.

SoftDoc™ will get people to agree with you.

SoftDoc™

by LEARNING TOOLS

LEARNING TOOLS 686 Massachusetts Ave. Cambridge, MA 02139 Tel. (617) 864-8086

USUS News And Report

March, 1985 Number 13

Eliakim Willner Editor

The USUS News And Report is published by USUS, P.O. Box 1148, La Jolla, California, 92038. USUS News And Report is a direct benefit of membership in USUS. This USUS News And Report was produced entirely with p-System software tools.

Inquiries regarding membership in USUS should be sent to the Secretary at the aforementioned P.O. Box. Newsletter correspondence and advertising should be directed to the Editor.

Institutional Members

Apple Computer
Attn: Lance Saleme
20525 Mariani Ave
Cupertino, CA, 95014

Donnelly Marketing Information
Attn: Dr. Gary Meyer
13511 Washington Blvd.
Stanford, CT, 06902
(203) 357-8773

Haverford College
Attn: Ed Meyers,
Dir. of Academic Computing
Haverford, PA, 19041

Hayes Microcomputer Products
Attn: Dan McCutcheon
5835-A Peachtree Corners East
Norcross, Ga 30092
(404) 449-8791

Japan Business Automation
Myojo Bldg. 3-50-11
Sendagaya, Shibuya-ku
Tokyo, 151, JAPAN
03/404-2221

NASA-Ames Research Center
Attn: W. Crawford
M.S. 233-15
Moffett Field, CA 94035

NCR Corporation
Attn: Ian Kaplan
11010 Torreyana Rd.
San Diego, CA, 92117
(714) 452-1020

Pinnacle Systems
10410 Markison Rd
Dallas TX 75238
(214)340-4941

SofTech Microsystems
Attn: Sharon Koehler
16885 W. Bernardo Drive
San Diego, CA, 92127
(619) 451-1230

Stride Micro Inc.
4905 Energy Way
Reno, NV, 89502
(702) 322-6868

Texas Instruments
Attn: B. Peterson
PO Box 226015
Dallas, TX, 75266

Xycom, Inc.
Attn: John Van Roekel
750 N. Maple Rd.
Saline, MI, 48176
(313) 429-4971

Officers, Board and Electronic Mail Contacts

Officers

President
Jim Harvison
P.O. Box 3277
Silver Spring, MD 20901
301/593-2994
Telemail: President
MUSUS: 70320,165

Executive Vice President
Temporarily vacant

Vice President, Member Services
Carl Van Dyke
SPM Support Services
10210 Iron Mill Road
Richmond, VA 23235
804/320-0144 (eve.)
804/320-2561 (days)
Telemail: CVanDyke
MUSUS: 70376,1435

Secretary
Michael Hartman
P.O. Box 3277
Silver Spring, MD 20901
301/445-1583
Telemail: Secretary
MUSUS: 73075,1171

Treasurer
Bart Thomas
First National Bank of Central Jersey
1 West Main Street
Somerville, NJ 08876
201/685-8395
Telemail: Treasurer
MUSUS: 76703,1031

Board Members

Eli Willner (Chairman)
Eliacomputer
1510 East 4th Street
Brooklyn, NY 11230
718/336-4109
Telemail: EWillner
MUSUS: 76703,500

Dennis Cohen
215 North Kenwood Street; #102
Glendale, CA 91206
818/956-8559
Telemail: DCohen
MUSUS: 71076,1377

Gary Pritchett
SofTech Microsystems
16885 West Bernardo Drive
San Diego, CA 92127
619/451-1230
Telemail: GPritchett

Carl Van Dyke
SPM Support Services
10210 Iron Mill Road
Richmond, VA 23235
804/320-0144 (eve.)
804/320-2561 (days)
Telemail: CVanDyke
MUSUS: 70376,1435

USUS(UK) also covering Europe

Chairman
Mark Woodman
The Open University
Faculty of Mathematics
Walton Hall
MILTON KEYNES MK7 6AA
England
(0908) 653187
(0908) 653678
Telemail: UKChair

Membership Secretary
Angela Grey
Stark Associates
Claremont Buildings
Salbrook Road
SALFORDS
Surrey RH1 5DY
England
(02934) 76747

Electronic Mail Contacts

MUSUS SYSOP
Bob Peterson
P.O. Box 1686
Plano, TX 75054
214/995-0618 (days)
Telemail: BPeterson/TI
MUSUS: 76703,532

Telemail ADMIN
(Contact Secretary for billing information and new accounts)
Jai Gopal Singh Khalsa
MicroStrategies
Box 2278
Vineyard Haven, MA 02568
Telemail: ADMIN
MUSUS: 72355,1015

Local Groups

USUS Washington/
Baltimore Local User's Group
Carl Van Dyke
SPM Support Services
10210 Iron Mill Road
Richmond, VA 23235
804/320-0144 (eve.)
804/320-2561 (days)
Telemail: CVanDyke
MUSUS: 70376,1435

SIG and Committee Chairmen

Advanced System Editor SIG
Samuel Bassett
34 Oakland Avenue
San Anselmo, CA 94960
415/454-7282
Telemail: SBassett
MUSUS: 71735,1776

Apple SIG
(Co-chairs)
Dennis Cohen
215 North Kenwood Street; #102
Glendale, CA 91206
818/956-8559
Telemail: DCohen
MUSUS: 71076,1377

John Stokes
9223 Skokomish Way NE; #2
Olympia, WA 98506
206/459-7598
Telemail: JPStokes
MUSUS: 70345,1256

Bart Thomas
First National Bank of Central Jersey
1 West Main Street
Somerville, NJ 08876
201/685-8395
Telemail: Treasurer
MUSUS: 76703,1031

Application Developers' SIG
Harry Baya
565 Broadway; Apt. 2H
Hastings-on-Hudson, NY 10706
914/478-4241
Telemail: HBaya
MUSUS: 72135,1667

By-Laws Committee
(Acting chair)
Bob Peterson
P.O. Box 1686
Plano, TX 75054
214/995-0618 (days)
Telemail: BPeterson/TI
MUSUS: 76703,532

Communications Committee
Bob Peterson
P.O. Box 1686
Plano, TX 75054
214/995-0618 (days)
Telemail: BPeterson/TI
MUSUS: 76703,532

Communications SIG
Bob Peterson
P.O. Box 1686
Plano, TX 75054
214/995-0618 (days)
Telemail: BPeterson/TI
MUSUS: 76703,532

DEC SIG
Henry Baumgarten
3325 Hillside Street
Lincoln, NE 68506
402/489-6441

Education SIG
Connie Gruber
SofTech Microsystems
16885 West Bernardo Drive
San Diego, CA 92127
619/451-1230
Telemail: CGruber

Graphics SIG
Alan Freiden
Information Systems, Inc.
3865 Wilson Boulevard; Suite 202
Arlington, VA 22203
703/522-8898

IBM PC/Compatibles SIG
(Co-chairs)
Harry Baya
565 Broadway; Apt. 2H
Hastings-on-Hudson, NY 10706
914/478-4241
Telemail: HBaya
MUSUS: 72135,1667

John Blasquez
P.O. Box 305
Walnut Creek, CA 94596
415/935-9295

Meetings Committee
Dan Merkling
Mentor
P.O. Box 11881
Salt Lake City, UT 84147
801/969-7041
Telemail: DMerkling

Modula-2 SIG
Dennis Cohen
215 North Kenwood Street; #102
Glendale, CA 91206
818/956-8559
Telemail: DCohen
MUSUS: 71076,1377

Pinnacle SIG
If interested in chairing this,
contact Michael Hartman, Secretary

Publications Committee
Editor
Eli Willner
Eliacomputer
1510 East 4th Street
Brooklyn, NY 11230
718/336-4109
718/336-4834
Telemail: EWillner
MUSUS: 76703,500

Assistant Editor
Arley Dealey
Pacific Systems Group
3615 Security
Garland, TX 75042
214/349-1515
Telemail: ADealey
MUSUS: 70130,177

Stride/Sage SIG
Bob Peterson
P.O. Box 1686
Plano, TX 75054
214/995-0618 (days)
Telemail: BPeterson/TI
MUSUS: 76703,532

Software Exchange Library
Distribution Chairman
Jon Bondy
P.O. Box 148
Ardmore, PA 19003
215/642-1057
Telemail: JBondy
MUSUS: 71545,2023

Review Chairman
Dennis Cohen
215 North Kenwood Street; #102
Glendale, CA 91206
818/957-4411 (days)
Telemail: DCohen
MUSUS: 71076,1377

Technical Issues Committee
Temporarily Vacant

Texas Instruments SIG
Danny Cooper
1709 Fairfield
Plano, TX 75074
MUSUS: 70735,1122

USUS Archive
Archivist
David Ramsey
1510 S. Bascom Avenue; #8
Campbell, CA 95008
408/377-6297
Telemail: DRamsey
MUSUS: 70076,1161

Word Processing SIG
Samuel Bassett
34 Oakland Avenue
San Anselmo, CA 94960
415/454-7282
Telemail: SBassett
MUSUS: 71735,1776

Editorial

By: **Eliakim Willner**

So, now I'm an editor.

This issue of the Newsletter is, as usual, a bit late. I won't bore you with the reasons, but we expect future issues of the Newsletter to be more timely. For one thing, the jobs of editor and publisher have now been combined; my trusty new LaserJet is producing the camera-ready copy (with the able assistance of ScenicWriter/HP) and the job is being printed and mailed by an outfit local to me.

Issue number 14 should follow this one shortly. Due to the lag in publication that preceded this issue, I have plenty of material for number 14. But there won't be material after that unless you, USUS member, contribute!

What kind of submissions are we looking for? The aim is for contributions with substantial content. This does not mean that we are looking exclusively for advanced technical articles. We solicit articles on all levels. Let the contents of this issue be a guide as to the kind of submissions we are looking for.

We want articles on all versions of UCSD Pascal and the p-System. We are interested in pieces on all the languages the Pascal system supports. Share your discovery of "gotcha's" and interesting programming techniques. Share stories and bits of humor you think might be of interest to the USUS community.

Our present intent is to cut back on "fluff" content such as the Software Source Directory and the directories of the USUS Library disks. Not that these aren't important resources, but they are rather bulky and don't change dramatically from one issue of the Newsletter to another. Directories of new Library disks will always be published and the complete Library contents, as well as the complete SSD, will appear in the Newsletter on a periodic basis. Note that both the library contents and the SSD are available online at all times on MUSUS, the USUS Forum on the CompuServe Information Service (MUSUS has been referred to by some as the lifeblood of USUS).

Letters to the Editor are also welcome. Your letters will tell us whether or not the USUS News and Report is on the right track.

The preferred means for getting Newsletter contributions to me is via CompuServe or Telemail. Please send plain text; I will insert the appropriate Sprinter directives. I can also accept Newsletter contributions on Stride or IBM PC format diskettes, or "standard" 8" UCSD format diskettes. If you have absolutely no way of sending your submission in "soft" form then please send me a hardcopy. But be forewarned that, since I will probably have to type in hardcopy submissions myself, I will strongly favor those articles that arrive in softcopy form! My electronic and physical addresses appear in the USUS officer list in this issue.

We also welcome display and classified ads! Our ad rates are quite reasonable and result in very high quality leads, if you sell UCSD Pascal or p-System related software or hardware. Contact me for details.

I would like to express thanks to Erik Smith and Scenic Computer Systems Corporation for providing the ScenicWriter macros that were used for the production of the previous Newsletter; they made *my* job much simpler. Erik graciously made himself available over the phone for technical assistance as I was bringing myself up-to-speed with ScenicWriter. The p-System community can be proud that ScenicWriter -- which, in my personal opinion, is the best word-processing software on the market -- is a p-System product.

Finally, some last-minute notes that I didn't want to defer until the next Newsletter:

Granville Kirkup is compiling a UCSD Pascal Resource Directory of available p-System experts, consultants and contractors -- including companies and individuals. If you wish to be included, send the details, including your area(s) of specialization, to: Granville Kirkup, GK Microsystems Inc., PO Box 440412, Aurora, CO 80044. We intend to publish this list in the Newsletter.

There has been much confusion as to whether overseas USUS members were permitted to access the CompuServe Information Service, and thus MUSUS. Some overseas members reported being told that they were not eligible to use the system after it was determined that they were not residents of North America. I have been assured by a person in a position of authority at CompuServe that this is not the case. CompuServe welcomes subscribers from anywhere in the world (except for a short list

of countries such as Cuba, North Vietnam, etc.) provided they have an internationally recognized credit card for billing purposes (such as Visa or MasterCard). The CompuServe representative said that if anyone was removed from the system the action was taken in error.

Overseas USUS members may access CompuServe via Tymnet, Telenet or local PTTs. MUSUS is a rich and active resource; if any member possibly can get on, they should do so. Overseas members who still encounter difficulties may contact me and I will attempt to intercede through the aforementioned CompuServe official.

Speaking of CompuServe, USUS no longer sells CompuServe Starter Kits; a variety of distributors are selling them for less than we were able to. One such distributor is Best Products, Inc. Their price at the time of this writing was $27.97. Best Products may be reached at (800)221-BEST.

USUS is now undertaking a drive to increase the number of institutional members. Institutional membership in USUS costs $500/year and provides the institution with the benefits listed below:

- One free copy of the USUS mailing list on Cheshire labels
- Unlimited internal use of the USUS library software
- Five copies of each Newsletter issued during membership term
- A 10% discount on all Newsletter advertising
- A CompuServe Starter Kit, including five free hours of online time
- The right to establish a Telemail subnode for your organization
- Prominent mention as an institutional supporter of USUS in the front of each Newsletter

We believe that these benefits, which include a number that have never been offered before, make institutional membership in USUS a potent bargain. If you are a company that sells p-System software, USUS members represent the cream of your market -- the mailing list alone might very well repay the cost of institutional membership. So join USUS and help yourself as well as your fellow UCSD Pascal users. If you are already an institutional member, renew and continue your benefits. Contact Jim Harvison or Mike Hartman for more information.

The following information on TI 99/4A p-System availability status was supplied by Mike Hartman.

"Over the past year, USUS has received literally dozens of letters asking about the availability of both p-code cards and p-System software for the TI 99/4A. Many people were able to find the cards in the wake of the cancellation and just need the software, while others are looking for both. I'm afraid the news is not encouraging, but not hopeless, either. When TI cancelled the 99/4A, it sold all of its rights to the machine, hardware and software, to Triton Products Company. Given sufficient interest, Triton could manufacture and sell p-code cards, but they are not doing so now. The p-System software is a trickier problem. TI made certain modifications to the basic system sold by SofTech Microsystems in order for it to run on the 99/4A. It (and now Triton) had rights only to these modifications; SofTech owns the rights to the IV.x p-System itself, but has no rights to distribute TI's modifications. Thus, for someone to legally distribute the 99/4A p-System, an agreement must be reached between Triton and SofTech. This will happen only if Triton is convinced that there is a market for p-System software among 99/4A owners. So if you own a TI 99/4A and want to buy a p-code card or p-System software for the machine, I recommend you contact Triton directly and let them know. Their address is:

Triton Products Company
P.O. Box 8123
San Francisco, CA 94128
(800) 227-6900

"USUS is also actively pursuing this matter. With all of our efforts, there may be good news to report in the next newsletter."

Much of the material in this issue of the Newsletter was gathered by Ro Lutz-Nagey, who was forced to resign as editor for personal reasons.

Eli

Toronto Meeting Minutes

Publications Committee Meeting

The meeting was called to order shortly after 2:00 PM.

Issues:

USUS Board on Telemail:

A policy to allow the posting of any messages to the USUS board to be considered republishable was proposed. This policy would be superceded by any explicit copyright notice posted along with the message. After discussion it was agreed that all prospective articles would be submitted to the author(s) for formal release for publication.

Newsletter "Typesetting":

The committee passed a recommendation to the USUS Board of Directors to buy a laser printer for the USUS Sage computer to be used for the production of camera-ready copy for the newsletter.

SIGs and the Newsletter:

A recommendation will be forwarded to the USUS Board to add a new item to the definition of a SIG chairman's responsibilities. This would be that each SIG chair must assure that their SIG produces an article for each newsletter.

Advertising Rates:

The general feeling was that the newsletter advertising rates are too low. The committee also wants to offer some free advertisement space to institutional members.

MUSUS:

Who owns the copy from MUSUS? Can a regular column be added to the newsletter with the "Best of MUSUS"?

Discussion:

It is likely that Scenic will print the next newsletter.

Publishing dates need to be included in the newsletter. The pertinent dates are copy cutoff, ad cutoff, and date the newsletter is to be mailed.

Discussion occurred regarding the article proofing policies of the newsletter. There does not seem to be a written policy currently. There was concensus that a formal policy should be developed. Ground rules are that all authors will get a listing by MUSUS, Telemail, or U.S. Mail of the final copy of their article and will be requested to sign a release form for the article.

The issue of publishing program source code in the newsletter was discussed. It was determined that the membership should be surveyed for their preference.

Letters to the secretary were discussed and the development of a question and answer section of the newsletter was proposed. This will be added if possible.

The meeting was then adjourned.

Minutes of ASE Sig (Unapproved)

1. Chairman Samuel Bassett was not present and no permanent replacement could be found. He remains chairman by acclamation in absentia, and David Lewis agreed to be temporary secretary.

2. People wanted to know what the situation is with Volition Systems. Dennis Cohen and George Symons held forth. The relevant points for ASE users are: (a) send bug reports to Arley Dealey; (b) there seems to be no more funding of ongoing development work on ASE other than a maintenance update and IV.2 version (by Arley) and possibly versions for native-code 68000 and the Mac; (c) VS's new address is: 4490 Fanuel St #208, San Diego 92019, 619-270-6800.

3. Suggestions and concerns from the populace for ASE development, roughly in order of priority:

a. Provide for entry and display of non-printing characters such as those which invoke features of printers and terminals ("raw" mode and monitor mode).

b. Support ANSI terminals and the ANSI function key sequences on Televideo terminals.

c. Do an MS-DOS version. A rumor was propogated (originated?) that there is or will be a Turbo Pascal version by authors unknown (or at least inaudible). This evoked general skepticism on technical grounds.

d. Do windows. But, many believe that this would make a different editor and require a substantially different program, so we were just engaging in some idle free association. It is controversial at best.

4. Lots of people want to learn to use ASE macros but are insecure about cracking that nut. Several others mentioned Karpinski's sample (and useful) macros on MUSUS (and the diskette library?). Karpinski responds well to feedback and attention, it was said, so give him some.

Respectfully submitted
David J. Lewis
1984 October 31

Minutes of Modula-2 SIG

SIG was called to order by the Chairman Dennis Cohen at 6:20 PM

Attendees were: David Rhoads, Charlie Carman, Michael Hartman, Jim Harvison, Henry Baumgarten, Edward R. Powell, Chris Jewell, Grant Phipps, Howard LeVaux, Karl Pleger, Alex Kleider, Alan Tompkins, Bill Bonham, Marsha Roberts, Jim Ochs, David Lewis, Eric Hafler,

Arley Dealey, Dennis Cohen and Herman Euwema.

Agenda Items:

Standardization Efforts.
Availability
Programming Techniques for Keyboard Entry.
Library Disks
MoSys
Favorite Bugs
Modula-2 Benchmarks

Introductions were made around the room.

Dennis reminded the SIG that the Chairman is required to see that an article is submitted from the SIG every other other newsletter and strongly encouraged to submit an article for each issue.

Library Disks. Contributions have been made to the library but there may have been a problem with the submission form. Dennis will follow up to see that it gets into distribution. The submission contains the source to p-Shell, a SOS version of p-Shell and Libmap. Contributions are encouraged and should include documentation for the submission. Submissions should be sent to Dennis Cohen at 215 North Kenwood Street, #102, Glendale CA 91206.

Benchmarks. Dennis has run the Sieve with curious results. On the Apple /// it ran 10% slower than Pascal. On the Sage, faster. On the IBM it ran 8% slower than NCI's p-System pascal. Bill Bonham has been working on a Native Code version of Modula on the Sage which was about 1.7 seconds, the same program on the Stride ran in about 1.4 seconds. He heard of another implementation which took about 4.3 seconds. The system has been bootstrapped under IV.13 as a loader but runs outside of the p- System. Bill has written an assembler in Modula-2 and is pleased with its performance. This is a two pass assembler and handles around 7500 lines per minute on a SAGE and on a Stride at about 11,000 lines per minute out of RAMDISK. Chris Jewell reported on benchmarks and he will post the results of the benchmarks on MUSUS and Telemail.

Availability. Volition: Sage, IBM PC standard p-System and p- System under a MSDOS bubble, CPM, Apple // for 1.1 1.2 and 1.2 128K and for the Apple ///. Logitech: 8086 Native compiler for MSDOS and CPM86 and a cross compiler for the VAX. This is basicly the Lilith compiler which generages 8086/8088 code rather than m- Code. MRI: PC, MAC, LISA, and Apple II. This version is also essentially derived from the Lilith compiler.

Dennis adjourned the meeting at 7:10 PM

The second meeting of the Modula-2 SIG was convened on Sunday at 10:17 AM by Dennis Cohen.

Attendees were: Dennis Cohen, Chris Lee, Bill Bonham, Verlene Bonham, Randy Bush, Kathryn Hjorleifson, Henry Baumgarten, Eric Hafler, Herman Euwema, David Rhoads, Alex Kleider, Jim Ochs, David Lewis, Chuck Emery, Randy Bush, Chris Jewell, Mable Roberts and Jim Harvison.

Volition Systems new address is: Volition Systems 4490 Fanuel St #208 San Diego, CA 92109 (619) 270-6800

Agenda Items:

More on product availability
More on the progress of standardization
Programming techniques for keyboard entry
MoSys
Favorite bugs

More on product availability. Volition: 68000 native code will move from alpha to beta test and will be released on the Sage. Macintosh product will be released 4th quarter of 1984, Cross compiling can occur from the Sage, files can be Inhaled and Exhaled, Bubble Environment, will run on 512K Mac with separate code and data, pricing should be less than current product pricing, native code cross compilation will be possible from the Sage to the Macintosh. Volition is alive and well and living in San Diego.

More on Standardization. Two main threads. Ad hoc committee has been meeting around the world for the past year and the definitions for the standard library modules have converged. Participants have been ETH Zurich, Volition, Logitech, OCE (formerly Diser). Structure of the library is most of what you would expect, the difficulty has been files. There are module files, texts, binary. Philosophy is you open the file with FILES, if you then touch it with TEXTS you don't then use BINARY. There is a module POS. No constants exported, no variables exported just types of procedures.

The British Standards Institute will meet next month to begin the definition Standards of Modula-2. Potential problems or discussion items are: multi-dimensional arrays, exponentiation operator, addition of more types in system "Byte", "AddressInc", "SizeInBytes", "Set of Char".

Programming Techniques for Keyboard Entry. How are people handling keyboard entry with error handling. Recommendation to read a character at a time and handle it directly.

MoSys. Is in development by TDI. Cost will be about $500 for object. Four times that for Source. Being developed for Sage and Pinnacle. This is an operating system developed in Modula-2. This operating system will provide full support for 32 bit systems and improved user interface over Unix.

The SIG was adjourned at 11:04 AM.

Modula-2 newsletter is available from MODUS. Write to:

Modus
PO Box 51778
Palo Alto, CA 94393
Membership is $25.

Modus' next meeting will be coordinated with the Edinburough meeting of USUS(UK).

The meeting was adjourned at 11:15 PM by Dennis

Apple SIG Meeting

Bart Thomas opened the meeting at 10:11 AM.

Attendees:

Bart Thomas, John Stokes, Dennis Cohen (co-chairs), David Lewis, Jon Bondy, Herman Euwema, Alan Tompkins, Bob Moody, Ron Stein, Henry Baumgarten, Jim Harvison, Harry Baya, Chris Jewell, Ro Lutz-Nagey, Michael Hartman, Carl Van Dyke, Eric Hafler, Glenn Hoffman, and Peter Williams.

In the library room [at Toronto] there were the fixes to allow Apple FORTRAN to run with Pascal 1.2 (does not use 128K) and a fix to make the ProFile driver work correctly with the 128K system. Bart also reported that a bug report for Tom Swan's Pascal Data Base System (PDBS) and the fix for the bug was available. The bug was related to large files of over 5000 records.

Dennis Cohen reported on Tom Swan's PDBS which is a relational database system and judged to be very good. It is published as a book containing source code and commentary, with or without program disks.

Introductions were made around the room.

Bart reminded the SIG that as a group the SIG is responsible for producing at least one article every other newsletter and that it was desirable to have an article every issue.

Cards to speed up the Apple // were discussed. Many people indicated they had Titan's Accelerator II which seems to work well as long as you do not have any DMA devices. There was one report that there seems to be a timing problem with the new Accelerator //e and some Apple II+s. This does not occur with all Accelerator //es or with all Apple IIs, and can be resolved by mix-and-match. One person reported that he had the SpeeDemon from MCT, which seems to work well also.

Randy Hyde's book "p-Source" was discussed, which gives mostly good information on program optimization.

Bart reminded the SIG that if you are moving to Apple Pascal 1.2, there are several files on MUSUS which will be useful.

In general, the upgrade to 1.2 has been slow in being delivered from Apple. Concern was expressed that Apple dealers have not been doing their job in informing users of the upgrade and follow-up/add-on products to 1.2 such as SANE (the Standard Apple Numerics Environment).

Comment was made that Apple should provided a new consolidated set of manuals for Apple // Pascal rather than the collection of updates and addenda which are very cumbersome and discouraging to new users.

The SIG was very concerned that the Apple Liaison to USUS, Lance Saleme, who had been invited to the meeting did not attend. In the past Apple has been well represented, and USUS should notify Apple that their active participation is appreciated and beneficial, and request that it continue.

The p-System on the Apple Macintosh was discussed. It is now available from SofTech Microsystems for $195.00. It provides access to the QuickDraw routines and the mouse. A UCSD Pascal compiler which runs under the Mac's operating system is expected shortly from SofTech.

Bart adjourned the meeting at 11:52 AM.

Library Committee Meeting

Meeting called to order by Jon Bondy shortly after 12:00 noon.

Attendees:

Chairman Jon Bondy, Mike Hartman, Jim Harvison, Harry Baya, Arley Dealey, Al Tompkins, and Herman Euwema.

Many of the distributors are having problems: not providing reports, failing to distribute, disappearing act. Six new volumes of the library have been submitted recently.

Jon's experience has been to distribute 400 disks in the first quarter. The breakdown of the types of disks distributed is: 8-inch disks virtually none, IBM and Sage the most, Apple next.

The library has some very good programs. Reviews of entire library disks or programs on the disks are needed and should be done. Members should be encouraged to review, submit bug reports and fixes, and indicate their general experience with the library. Harry Baya committed to doing the first library review.

An alternative to the current distribution approach was suggested with the centralization of ordering. This was discussed. It was agreed that the Secretary would become a distributor for the Sage and Apple formats.

SofTech wants to collect application development tools for public-domain distribution.

Major Vendor Panel

Panel Members:

SofTech Microsystems (SMS) was represented by the SMS Liaison to USUS, Connie Gruber. TDI was represented by Mark Wiggin. NCI was represented by Stephen Pickett. Chris Jewell, formerly of Apple Computer, agreed to try to respond to Apple questions. (Lance Saleme, the Apple Computer Liaison to USUS, did not attend.)

1. Will the December release of the SofTech Macintosh p-system support the 512K Mac? Answer: Yes.

2. Please clarify the SofTech Mac products. Answer: The current Mac p-System product will support the 512K Mac in December. The new product (stand-alone compiler) will support the 512K Mac at release. The upgrade for the current product will be approximately $50.

3. What computers does TDI support the p-System for? Answer: Acorn, BBC, Commodore 64, and others [not recorded].

4. In the past SMS has been mostly marketing to hardware developers; now it seems that there has been a shift to individual software developers. Is this true? Answer: The OEM channel has changed and SMS is directing their efforts to the retail market but has not settled on a particular approach.

5. What is the level of cooperation between NCI and SMS and can we expect compatibility? Answer: NCI: Wishes to maintain compatibility. They expected to have received IV.2x by now but have not. SMS: Beta-test sites have been selected, and as soon as the product is released from beta-test then licensees shall receive it.

6. Is NCI one of the beta-test sites? Answer: SMS: No.

7. With the upcoming stand-alone Pascal compiler for the Mac OS, can we expect to see other operating systems supported by SofTech? Answer: The Mac system was a particular response to a market need. If there seems to be a need in other areas they will consider it.

8. What are TDI's markets? Answer: Stride, Pinnacle, Victor 9000, p-System, adaptations to other systems.

9. Who are TDI's major competitors in the European market? Answer: There does not seem to be any competition for the 68000 machines. There are no other major p-System vendors in the UK.

10. Is the Advanced File System (AFS) dead? Is Liaison not being pushed? Answer: AFS is not on a schedule to be a product. Regarding Liaison, SMS is very pleased that Liaison is included on the Stride. SMS is very interested in talking with any manufacturer who is interested, but they are not actively seeking machine support. Stride is pleased to have the local-area network supported and plans to continue support for it. SMS has licensed the product to Olivetti. The license is not specific to a piece of hardware.

11. Will we see COBOL on the p-System? Answer: SMS has no plans. Stride plans to bring up a version of RM/COS on the multi-user Stride.

12. Stride/Sage has been shipping the p-System with every box. Does Stride think that this will change in the future? Answer: Stride has every machine licensed for the p-System. About 10% of the machines do not have the p-System actually sent with them. Idris has been doing very well. If they are to move away from the p-System, that will depend on what users want. UNIX Version V should have greater popularity than the current Idris.

13. Tell us about MOSYS. Answer: TDI: MOSYS is a native-code 68000 operating system running on the Sage. It is written entirely in Modula-2. It is not available at this time.

14. What is the current state of IV.x targeted for the Apple //e with the extended memory? Answer: SMS: This is not at the top of the list for development.

15. Does MOSYS take direct advantage of the address space of the 68000 and support files larger than 16 megabytes? Answer: Yes.

16. What are the prospects for the distribution of MOSYS in the U.S.? Answer: TDI would like to do so.

17. Does SofTech have a policy on helping p-System developers, and what can the developer expect? Answer: SMS has an intent to do so but there is no policy. Taking the application approach is one method to assist along with the revised licensing plans. SMS will be announcing a p-System database package for the Macintosh. SMS encourages developers to contact SMS to discuss ways to encourage mutually beneficial arrangements.

18. Does SMS have any plans to encorporate "Caller", developed by NCI, with their system? Answer: No, not at this time.

19. Are there any plans for an 8087 code generator and any plans to generate additional native code over what is currently done? Answer: NCI is licensed for the 8087 native-code generator but they have not seen it. SMS said they have just released the 8087 native-code generator and have improvements for the current 8086 code generator.

20. Is NCI licensed for IV.2 or just IV.13? Answer: NCI thinks that IV.2 is considered an upgrade and therefore they should be licensed for it. SMS confirmed that this is the case.

21. The MS-DOS hosted p-System is an upgrade for additional cost.

22. Have there been any changes to SMS maintenance policies? Answer: Yes. Before there was ongoing free support for any who made purchases directly from SMS and not much support was available for anyone else. Now for $75 any version IV.x owner can get 12-month support and upgrade options.

23. Are there any improvements anticipated for the 68000 native-code generation? Answer: No plans at this time.

24. Are there any plans for NCI to support any processors other than the 8086/8088? Answer: Yes, the 80286 (IBM PC/AT).

25. Are there any plans for other native-code generators, in particular the 6502? Answer: No. The 6502 p-code to native code expansion factor is about 1 to 6. Other people have attempted to do a 6502 native-code generator and have reached the same conclusion that this makes the use of native code infeasible.

26. Does NCI have any plans to produce p-System adaptations outside the Intel chip family? Answer: Not this year.

27. What is the status of the VAX implementation of the p-System? Answer: It has been alpha-tested but is not close to being released. [At the time this Newsletter went to press the VAX implementation was being sold by TDI.] SofTech and TDI both have rights to market it.

SofTech Presentation Notes

Connie Gruber - Manager of Marketing Support
Designer Series, new marketing strategy

They are trying to increase the marketability of the environment. Includes: Editor, Filer, Utils, Run time system, services, the Language Compilers are optional, as are the Tools.

Choice of UCSD Pascal, Fortran-77, BASIC, Assembler.

Optional Tools: Symbolic Debugger, Native Code Generator, Insight Window Designer, EDVANCE and KSAM.

Hosted Systems:

p-System and MS-DOS co exist; Applications can concurrently use MS-DOS & p-System files; p-System File Manager; p-system files on p-sys disks, p-sys virtual volumes in MS-DOS. All MS-DOS supported devices may use standard p-sys drivers, generic ASCII Console driver.

Macintosh: 50 MAC ROM routines, Quickdraw and Mouse. 68000 assembler wth Advanced Development Tool Kit.

There will be an article in the December Byte on the p-System. Have shipped 500 copies at this point.

Next product: Create UCSD Pascal programs using MAC Operating System. p-system not apparent. Includes: Mac Editor, UCSD Pascal Compiler Debugger and access to 500 MAC ROM routines. Available mid November.

UCSD Pascal Development System: Was $625 now $295 for p-System and MS-DOS $195 for p-System/Macintosh

New Upgrade Plan, Customer Support Policy: Customer support plan: SMS will provide first level support for the product 90 days free then $75 per year there after. Will get a 10% reduction of future products. Bug list for $15 each copy issued 4 times per year.

IV.21: Improved boot speed, automatic memory configuration, multiple code pools suported, external data areas supported, improved operating system performance.

Multiple Code pools give the ability to use extra memory with extra 64K code pools.

External Data Areas. Allows bringing extra data segments swapping with the Stack/Heap.

Set Command Added: Allows specifying what volumes the system files are on. Name work file. Improved work file handling.

Pascal procedural parameters, Conformant arrays, compatible with all IV.1 releases. Available late October. Softech still has goal to maintain source code compatibility.

Performance Tuning Tools: Have provided a Time Profiler Occurrence and Time monitor and Fault Analyzer to observe segment, stack, heap and pool faults.

Jack2: The Easy-to-use, integrated business package. Four tools in one: Word Processor, Spreadsheet, Graphs, Data Base. Perfect for Basic Business Needs. SMS has right for all machines other than Apple and IBM. With the Jack2 developers filing chapter 11 there are negotiations for Apple and IBM. Cannot be used currently on a non video maped machine. SMS is interested in acquiring p-System products to market.

Educational Software: Learning environments for UCSD Pascal/FORTRAN-77. Create instructional programs up to 2000 Lines and 6 units. Teach beginning programming courses. Two options; unit purchase per machine cost with number of machines: $150 per copy. Second plan; license, cost is based on number of students and sites. The cost starts at $795.

Standard product for educational software. Full-featured UCSD Pascal or FORTRAN-77 Compiler. Teach beginning through advanced couses. Develop programs for instructional delivery. Teach/develop using both languages.

IBM Apprentice program. SMS Pascal and FORTRAN are the two languages included in the program.

Minutes Old Board

Bob Peterson called the meeting to order at 8:23 AM on Sunday

Board members present were Bob Peterson, Randy Bush, Arley Dealey.

Agenda

Election Committee Report

The results of the election were:
551 Ballots returned

Dennis Cohen	435
Carl Van Dyke	373
Herman Euwema	189
Ro Lutz-Nagey	355
Stephen Pickett	58
Gary Pritchett	373
Eli Willner	551
R.C. Carroll	5
Steve Pillot	1
Disincorporation	5

Less than 10 invalid proxies 4 people at meeting substituted new proxy for old. Post card worked well, Secretary should prepare two labels one for the post card and one for the envelope. These should be prepared two up for correct posting. Ballots came in at between 5 and 10 per day. 2 ballots authorized Bob

Peterson to vote them as they see fit.

Randy Bush moved to adjourn. Seconded and Passed.

The meeting was adjourned at 8:29 AM

Minutes New Board

The meeting was called to order at 12:23 PM by Ro Lutz-Nagey on Sunday 10/14/84.

Present were: Ro Lutz-Nagey, Carl Van Dyke, Dennis Cohen, Eli Willner, Gary Pritchett

Agenda:

Officers
Telemail Acct for ACD ect.
ByLaws Comm
Meeting Comm
List of Objectives
Vendor Relations
Treas. Proposal
Library Proposal
Newsletter/Member Services

Eli Willner was elected chairman of the Board.

Candidates for Positions:

President: Jim Harvison, Herman Ewema.
Executive VP: John Stokes.
VP/Member Services: Carl Van Dyke.
Secretary: Mike Hartman.
Treasurer: Bart Thomas.

It was moved to appoint Jim Harvison, John Stokes, Carl Van Dyke, Mike Hartman, Bart Thomas. Seconded and passed without opposition.

Dennis Cohen moved to affirm the creation of a standing communications committee with the duties as assigned by the executive committee (Duties being to oversee the provision of Telemail and MUSUS services). Seconded and Passed without opposition.

Bob Peterson was appointed the chair of the bylaws committee.

George Symons was appointed the chair of the meetings committee.

All officers and board members shall be provided copies of USUS corporation documents and bylaws by the secretary.

Eli moved that the Board will support activities which will strengthen the relationships and participation of p-System vendors with USUS. Seconded and approved.

Dennis Cohen moved to reaffirm the position of USUS to be vendor independent. Seconded. Passed.

Carl Van Dyke moved to adjourn the meeting. Seconded. Defeated.

Ro Lutz-Nagey moved to elect Dennis Cohen as Vice-Chairman of the board. Seconded. Approved.

Ro Lutz-Nagey moved to adjourn the meeting. Seconded. Approved.

The meeting was adjourned at 12:55 PM.

Minutes Old Executive Committee

The meeting was called to order at 4:05 PM on Saturday October 12 by the President, George Symons.

The Attendies were: Dennis Cohen, John Stokes, Bart Thomas, Ro Lutz-Nagey, Arley Dealey, Chris Jewell, Carl Van Dyke, Alex Kleider, Bob Peterson, Jon Bondy, Jim Ochs, Herman Euwemma, Edward R. Powell, Michael Hartman, George Symons, Verlene Joyce Bonham, Chuck Emery and Jim Harvison

Agenda Items:

1. Treasurer's report
2. Secretary's report
3. Laser Printer
4. SIG Chair
5. Advertising Rates
6. USUS(UK) news
7. Organization Structure
8. Telemail Publication Policy
9. Membership Renewals

Members of the Exec Comm are Officers, SIG and Committee Chairs.

Jon Bondy requested that in the future, all Officers Reports and commitee agendas be submitted via Telemail prior to the meeting.

Treasurer's report: Jon Bondy requested that the Treasurer's report include data for the year.

Telemail receivables are a problem. We were slow in billing and people slow in paying. There are about 100 Telemail users. Carl Van Dyke suggested that we include an amount for bad debts. Is there a breakdown for the cost of Telemail for the organization? George estimated that it was currently costing about $500 per month. In the future an appropriate cost should be $300 per month.

Secretary's report: The Secretary's report included discussion of the new members and renewal rate.

Month	Renewals	New Members (Domestic)	New Members (Foreign)
November 83*	243	94	4
December	94	75	6
January 84#	71	72	2
February	7	40	3
March	11	70	8
April	4	43	1
May	27	73	8
June	51	54	2
July	18	66	7
August	17	61	5
September	6	41	6
October	0	13	1

=====	====	====
549	702	53

*Corporate membership renewal received
#New Corporate Member

Dennis Cohen moved that the membership secretary request renewals as soon as possible. Seconded. Motion Passed.

Bob Peterson moved that members be credited with 12 months of membership from the date they have renewed. Seconded. Passed without opposition.

USUS(UK) news items: Next conference around April 11 in Edinburough. USUS(UK) has now become USUS(UK) Limited. Current Chairman Mark Woodman is on TeleMail (Address: UKCHAIR) and fairly active there. Formal communications with USUS will begin. Only problem has been the lack of Newsletters #11 and #12. Would like to get #13 out as soon as possible. Want to know how much it will cost to print 500 issues for USUS(UK).

Organization Structure: Arley suggested that a communication committee be formed to address the oversight of the Telemail and MUSUS on Compuserve. Ro Lutz-Nagey moved that a standing Communications committee be formed. Seconded. Discussion: Committee would have oversight of the Electronic communications services. Passed Unopposed. Bob Peterson was appointed chair.

Laser Printer: Currently $1,500 per newsletter issue for typesetting cost. Ro requested the consensus of the executive committee for acquiring the printer. Discussion: Bob suggested that the individual using the printer may wish to purchase the printer and lease use to the organization. Dennis suggested that any decision should be delayed pending expected announcements in new technology printers. The consus of the commitee was that they would consider a proposal to purchase or lease back a printer.

Advertising Rates: Ro requested that rates be raised. A full page black and white add costs approx $250. Bart moved that the current advertising rates be raised 25% not before issue 14. Seconded and Passed.

Jon moved that a 10% commission be paid on new advertisements obtained. Seconded. Motion was withdrawn.

Bart moved that the executive committee gives its approval in principle for payment of commissions for nes advertisements obtained for the newsletter. Seconded and Approved.

Telemail Board: Ro moved that any messages posted to the USUS board on Telemail would be considered available for publishing in USUS News unless their is a disclaimer included in the message otherwise. Seconded. Motion was withdrawn.

Bob Peterson moved that every SIG chair see that a article be prepaired for the newsletter at least every other newsletter and recommend that it be prepared for every newsletter to begin with issue #14 and further a chair not meeting this requirement may be removed from office. Seconded. Passed.

The meeting was adjourned by the President at 5:43 PM.

Minutes New Executive Committee

Attendees:

Jim Harvison, President
John Stokes, Executive VP and Apple SIG co-chair
Carl Van Dyke, VP Member Services and Board member
Michael Hartman, Secretary
Bart Thomas, Treasurer and Apple SIG co-chair
Henry Baumgarten, DEC SIG chair
Harry Baya, IBM SIG co-chair and Applications Developer's
Dennis Cohen, Board member, Apple SIG co-chair, and Modula
Bob Peterson, Communications and By-Laws Committees, Commu
George Symons, Meetings Committee chair
Ro Lutz-Nagey, Board member and Editor
Gary Pritchett, Board member
Eli Willner, Board chairman
Arley Dealey
Eric Hafler
Edward R. Powell

Agenda:

Secretary/Treasurer Staff Proposal
Library Proposal for Next Meeting
TI 99/4A Users' Problems
SIG Chair Reports
Committee Reports

Jim Harvison, President, called the meeting to order at 2:20 PM.

Secretary/Treasurer Staff Proposal:

A written description of a new proposal by Jim Harvison on paid clerical help for the Secretary and Treasurer was distributed to committee members. It recommended the creation of two salaried positions: Membership Secretary and Correspondence Secretary/Treasurer. The Membership Secretary would be responsible for maintaining the USUS membership database and for handling all correspondence on membership status and newsletter receipt. The Correspondence Secretary/Treasurer would maintain the USUS official correspondence records, prepare and mail all technical correspondence drafted by the Secretary, and also handle all the clerical duties of the Treasurer (except for Telemail billings). The continuance and salary of both positions would be reviewed at each semi-annual general meeting. This proposal is made in order to assure continuous, smooth maintenance of the membership database; to provide a single contact point on membership questions for USUS officers, members, and potential members; and to ensure compilation of complete records of USUS expenditures and receivables. Bart Thomas moved that the proposal be adopted.

A discussion of the responsibilities of the Treasurer was started by Eli Willner; he pointed out that the Treasurer would have the check-signing responsibilities; the Secretary would do only the administrative jobs and some of the clerical. Some

other officials do have check-signing authority; Jim said he (as President) would not want it here. George questioned whether there would be additional costs. It seemed to those present that they would be similar to those currently incurred for these tasks. Bob Peterson raised the issue of whether USUS was effectively hiring someone, whether such a thing should be done, and whether USUS should simply hire a full-time administrator to handle things. Arley Dealey said he was not currently philosophically opposed to the proposal offered, but practical matters such as cash-flow might be a problem. Ro Lutz-Nagey felt the proposal should have been brought to the board, as it involves substantial committment of funds. George Symons and Bob (former officers and board members) said the Executive Committee could commit the funds with the concurrence of the board. Ro was also concerned about the communications overhead of two people, instead of one doing it all. He felt that due to the cash flow the positions should be kept as hourly rate, not salaried. Harry Baya suggested the amounts of money proposed become upper limits, and that USUS get a contracting agreement written.

Bob reported that Jerry Pournelle has told him that L-5 and other societies hire retired military persons to do administrative duties. These people don't need high salary (have pension income) and have other advantages. Bob moved that the motion be tabled (Bart's motion to accept proposal in full). Bart seconded. Discussion following indicated that the proposal should be studied, and a decision on it reached on at the next interim meeting. The motion to table passed without opposition. George offered to post on Telemail an article on what constitutes an independent contractor.

Library Proposal:

Ro brought up a library proposal for quick mention now, to be discussed further on Telemail. He wants to improve visibility of the library in the Pascal community at-large by preparing a sample diskette of selected USUS Library software. This requires permission of its authors to move those programs into the public domain. The disk would be distributed by selected microcomputer user groups. Ro moved this proposal. Bob seconded it. Bart moved to table the motion; no second. Eli said the software is specific to the p-System; p- System groups already have some of it, others will adapt it to their flavor of Pascal and not help USUS. George said we should attend to the current, growing membership, not spend energy on others at this time. Some thought this is a library committee issue; Mike Hartman said the library is a key service and incentive for joining USUS, and its distribution is of interest to all. Ro moved to table the proposal to the Interim Meeting; Bart seconded. Motion to table passed unopposed.

TI 99/4A Users' Problems:

Jim says there is a large, growing segment of TI 99/4A owners in USUS who need basic p-System software. A number of them have written letters to him as Secretary. Many have p-System cards (bought in close-out sales), but no software for them. Neither TI nor SofTech is or can currently distribute the software. To make the p-System run on the 99/4A, TI had to strip down the system. TI sold all its 99/4A rights, including these software modifications, to Triton Products Company. SofTech can distribute only the basic, unmodified software. George moved that the Executive Committee direct the VP for Member Services to pursue a solution to this TI problem, and in the interim act as vendor liaison to Triton. Bob seconded. The motion passed unopposed.

SIG Chair Reports:

TI and Word Processing SIGs did not meet this time. Application Developer's tried but didn't. DEC SIG didn't either partly due to Eli's schedule; Henry Baumgarten will chair DEC now due to this and to Eli's business involvement with the p-System for DEC machines. Carl Van Dyke asked who determines the SIG scheduling, and stated that the IBM SIG was hurt by being held at the end of a long day. Bob reported the formation of a new Education SIG; Connie Gruber may be the new chairman (this is the 3rd incarnation of the SIG). Reports from other SIGs which did meet, including ASE and Stride (formerly Sage), were not received due to the shuffling of the meeting time. Full reports on the Apple and Modula-2 SIGs appear elsewhere in this issue.

Committee Reports:

The Meeting Committee reported that the next Interim meeting will be held on February 9-10, 1985 in conjunction with Stride Faire (Technical Issues Committee will meet on the 9th). The next general meeting will be in Dallas [since changed to Salt Lake City] in late April or early May. The exact date will be known in 2 weeks when hotel arrangements have been firmed up. The Meeting Committee is looking for suggestions on the following interim and general meeting sites. A suggestion was made that the fall 1985 general meeting be in Baltimore or Washington.

There were 68 paid attendees at this USUS meeting, 38 of whom pre-registered. There were $3300 Canadian taken in from registration, library disks, and 20-30 newsletters. Forty blank disks were sold at cost. Meeting attendees had 30 rooms in the hotel, yielding the mid-level rate for the conference rooms. This saves $600. The meeting cost about $2000 US, so that makes about $540 US ($700 Canadian) in profit. Bob points out that some people stayed four nights, not two, which may help us with the hotel fees.

The library committee met. The President or Secretary will be added as a USUS institutional distributor for Sage, Apple, and IBM formats. Also, library prices for certain formats will be standardized.

Arley reported that the Publications Committee met, and boasted a big turnout. He commended the new editor, Ro, for his work. The development of a newsletter staff to assist the editor was suggested. Arley will be the first assistant editor.

Eric Hafler asked about the availability of mailing lists for geographic areas to assist in forming local users' groups (LUGs). He was told to write the Secretary; mailing lists will be provided at no cost for this use. Jim said USUS might also underwrite the cost of the first mailing to organize an area's LUG.

Arley reported that a Communications Committee was formed with Bob as its chairman. The committee will handle the MUSUS and Telemail functions of USUS. Carl raised the issue of unpaid Telemail accounts; we need to recover these expenses through prompt billing and aggressive collection. Carl made a motion to enforce the policy of terminating access to Telemail for accounts for which payment has not been received 30 days after invoicing. This may be waived upon contact between the Treasurer and the accountee. No second was offered since this is current USUS policy. Arley believes Telemail collection is progressing and action on delinquent accounts is being taken. He was not clear on whether USUS policy is suspension or termination of delinquent accounts.

Bob Peterson agreed to serve as acting chair of the Bylaws SIG. A permanent chair will be sought.

Carl moved to adjourn; Eli seconded. Passed unopposed at 3:15PM.

Notes on Expert Users' Panel

The Panel began at 11:21 AM on Sunday.

Panel members were: Chris Jewell, Gary Pritchett, Barry Demchak, Randy Bush.

Randy Bush reported on what he observed on a recent trip to Japan. Observations were: 1965 approach, COBOL, FORTRAN and BASIC, No programming teams, one or two programmers working on a project, almost no penetration of Pascal. When you talk to or about MITA regarding the fifth generation and how the software technology be applied to the hardware technology they indicate they recognize the problem but have no solution. Low numbers of computers per student for schools, only "poor" students use computers for remedial work. Cooperation does not seem to be as extensive as we are led to believe. IBM is predominant mainframe vendor in Japan. Prolog is the language being used for the fifth generation. Randy will be writing a paper for US publication as well as Japanese publication.

1. Are the conformant arrays offered by SofTech in IV.2x in compliance with the ISO standard and if not, why not? A: There was an intent to make them conform and if they don't, reason not known.

2. Can the way procedural parameters are handled be discused? A: No, not enough information to respond to the question.

3. What is a conformant array? A: Allows writing routines that do not need to know the size of the array.

4. Initialization does not seem to work with OSI p-System. A: Initialization was added with Version II.1, the problem with is II.0.

5. Has a document on how multiple units are initialized been made available? A: Has not been formally released for public distribution. How initialize is handled is likely to change from version to version.

6. Is it possible to implement a "SideKick" like capability in Version IV and if not is there an intention to provide it? A: Not a current feature of the p-System. Could it be done?

7. Can a process that is waiting be swapped out of the code pool? A: Yes. Randy suggested that an application could be substituted for the System spooler to similate what the Borland "SideKick" product does.

8. Are there any plans to allow the user to install units in the operating system that will be initialized by the operating system at boot time? A: No such plans.

9. If gotoxy has an initialization section will it get called? A: Units in SYSTEM.PASCAL do not have initialization code invoked.

10. Please describe the way IV.2x supports data beyond the normal stack/heap data space. A: A set of units will be provided to allow the user to find out how much unused memory is available and allow moving data to and from the stack heap.

11. ? A: When a data segment is allocated it is allocated as a memlocked code pool segment (it is not position locked).

12. What is being done to fix the bugs in the chaining process? A: Priorities are set by user requests to fix problems. There have been few requests to fix the chaining bugs.

13. On a Apple with 1.2 is it possible to use an interrupt-driven communications program with a mouse active at the same time? A: Chris could not see why it would not work.

14. In reference to Apple Pascal, is CALLER relevant? A: There is a version of caller for Version II.0 in the library which should work with Apple Pascal.

15. How can {$U-} be used? A: There is a Unit on MUSUS done by Arley Dealey which allows use of the features of {$U-} in relatively safe ways.

Data Flow Computing

By: Tom DeMarco, Atlantic Systems Guild

From the October 1983 USUS Meeting

Keynote Lecture

First, I want to say I am very pleased to be here. I've followed this organization, and the system that it is concerned with, for all this time, and I have a great deal of fondness for what's going on here. I want to thank Winsor Brown, Randy Bush and Bob Peterson for their very kind invitation.

There is a little problem about being a keynote speaker: I guess one is supposed to hit the 'key' notes. The keynote in this gathering has to do with UCSD p-System, its history and its evolution. Unfortunately, Jeff Clark hit that keynote exactly on the center [in a "commercial" presentation given earlier] and you held his feet to the fire. Not only that, but you have a very long history of doing this. For instance, any of you that were in Dallas a year ago, or the meeting before that when Doug Ross had the temerity to turn up, saw that he had his feet held to the fire.

Therefore, I decided that, given that everybody in this room would know more about the p-System proper than I do, despite my five years use of it, I decided not to hit exactly a keynote, but rather a gracenote. So what we are going to have here is a 'gracenote' presentation, on a subject that you will see is in many ways akin to what's been going on in the entire Pascal revolution; which has to do not just with a different way of computing, a different way of constructing software systems, but has to do with extending the already different way of thinking about software that was started some 15 years ago with the block-structure language revolution. That is the topic of data flow computing. Something which has enormous advantage to me, since I know more about it than you do. So that you can't hold my feet to the fire too severely.

What I want to do is begin by talking about paradigms... I'll talk about paradigms for systems construction, one of which is what I call the data flow paradigm, and then give you four instances of that. Then I'll go on into the future, very slightly, to talk about what kinds of methods I think are right around the corner -- are already happening somewhere and are very likely to be happening to you sometime in the near future.

Having made use of this word paradigm I suppose I am obliged to define it. A paradigm is a sort of mental model. It turns out the human brain is not capable of thinking about all the things that are capable of being true. The human brain has certain built-in 'pigeon-holing' schemes, mental models if you will, that make it possible and easy to think about some things particularly well. For instance, clearly the human mind is a very adequately put together structure, internally organized to deal with things like language. Bertrand Russell, a philosopher, commented on this. He said, "The thing I don't understand is how I come to understand so much". And, in particular, referring to the idea of language. He concluded that the brain was very precisely organized so things clicked into place into subsequent language. Other subjects, art, for instance, and music... we've have much more difficulty clicking things into place, at least some of us do.

Another story about paradigms: in the voyage of the Beagle, Charles Darwin wrote down that the natives of Tiera Del Fuego were incapable of seeing the Beagle. The Beagle was a 94 foot brigantine with four or three masts, and it parked 400 yards off the shore and the natives couldn't see it. They looked out there, and didn't see it. They missed it. Yet the moment a little skiff put out behind it and came around, all of a sudden they all started shouting and pointing at the skiff, that they could see. They didn't have a paradigm, they didn't have a mental classification system for this great huge ship and therefore, they couldn't think about it very well.

One of the most evident paradigms of the last 10 years has been the top-down paradigm. We changed gears, around 1970, and started to think about things top-down. Some people think that they understand top-down implementation. For instances, they think a top-down implementation is just implementing with your 'top' down. That turns out not to be true [Laughter]. It has much more to do with this idea that's consistent with everything that you do when you do a Pascal program, of having a main program which invokes lower level functions, and those lower level functions and procedures invoke still lower level functions, so that the readability at the top is very evident in terms of the abstract components that make up the program. When you look at one of

those components, you get a little more detail. You look at the lowest level component and you get all the detail. That is the top-down paradigm.

I think the top-down paradigm is one of three paradigms that are kind of interesting if you build software. One of them is the control hierarchy concept. The top down paradigm was preceded in time by a control-flow paradigm. By that I mean, when we thought about software when I broke into this field in 1960, we put ourselves squarely in the position of a computer. When we thought about how a program would work you would say, "How does the computer think about this program, in particular, what is its order for dealing with things". First it does this, then it does that, then it does the third thing... and that's why so many programs were designed right up front with an initialization section. Today we don't tend to do that so much, because we don't think about things in the order that the computer thinks about them. That turns out to be a rather trivial order, because the first thing a computer deals with is very insignificant in terms of what the whole system of programs is trying to do.

So the control-flow, or the flow-chart paradigm, is one that persisted for a while. Up until about 1965, you saw people writing flow charts. Then all of a sudden everybody heaved a great sigh of relief and threw the damn things away. Well, not everybody... I was at an organization just recently that is selling a product which helps people keep their flow-charts up to date, and I said, "What people?". They said, "Well, we have this company that has tens of thousands of flow charts that they have got to keep up to date." I thought, "They not only missed the 1970's, they've seemed to have missed the 1960's, as well". They are the last people on earth that you could sell a copy of 'Auto Flow' to.

Today, we are much more inclined to deal with a 'control hierarchy'. That is a structure chart or HIPO diagram, that's the way we tend to think about systems, whether we actually draw it or not. The way you write code, with a main part at the top and the lower level routines which invokes still lower level routines, shows that you are making use of this control-hierarchy paradigm. An idea which, as I say, was rather new in 1965. It was actually brought to us by an obscure Dutch professor named Edsger Dijkstra. The man is obscure but not gentle. When he says something you know its been said. He has been characterized by a friend of mine as being 70% intellect and 30% bile. He had very strong feelings on the subject of control-flow paradigm. He felt that it was wrong and that procedures and languages that didn't support the new paradigm, the hierarchical top-down paradigm, were wrong. In fact, he was fond of torturing his undergraduates, that petitioned to be let in to his course, by asking the question; "Have you ever written a FORTRAN program?". If they said "Yes" he said "Out". They were not allowed to take his course because their minds had been irrevocably destroyed.

The idea of a control hierarchy, here I am going to be using some exhibits prepared by some of my fellow guild members (they are called Guildoons) James and Susan Robertson, who are part of our London office. Here you see a hierarchy in graphic form. The idea of this is just the graphic way to say that this module at the top invokes these three subordinate modules, has no visibility to anything beneath them. These subordinates invoke these two or, in this case, those four. So you can imagine what the code would look like. The code inside here would have three other modules, external modules, that it invoked. And this one would have four other ones that it invoked. The items shown along the invocation path are passed parameters. In the world of Pascal, the idea of passing parameters seems only normal, it's no big surprise. In the world of COBOL, when you introduce this idea, it seems like you are shaking the world apart. But I think for most of you the control hierarchy has become something fairly well understood and fairly universally endorsed.

Well, I think that this is not the final paradigm. I think this took us very nicely through the 1970's and into the 1980's, but I don't think it's going to be the all-important paradigm, the way to think about software construction toward the end of the 1980's. I think that we are going to start thinking about things not so much in terms of what control does, what the stream of consciousness of the computer does, but we are going to start thinking about it in terms of what the data does.

In order to introduce that idea, I'm going to show you a simple notation for presenting a system from the point of view of the data and then give you four rather widely spaced instances of the use of this thought pattern. As you will see, one of these is very close to some of the work that I've been doing, but the other three come out of left field, at least from my point of view.

The notation is a very simplistic notation, it simply represents a system in terms of subsystems.

The subsystems may be represented with a lower level partitioning in terms of sub-subsystems and the sub-subsystems in terms of sub-sub-subsystems, and so forth. Making use of the network graphic, which I will call a data-flow diagram. Of course, I presented it very abstractly, but if you keep in mind that each one of these connecting arrows is movement of a particular known data item. By known, I mean its internal composition must be known. And each one of these bubbles represents some sort of a transformation in which incoming data is transformed into outgoing data according to some set of rules. The rules are typically laid down by the user of a piece of software. Now you see what I mean by data flow diagram.

I am going to introduce a rule that each one of the data items ought to be described with a name. We ought to give a name to the data items, so that we could say, some place else, precisely what that data item consists of.

For instance, instead of using the abstract "S" that I have used here, lets call this data flow "Service Request", and of course we are going to go back and read each one of the other data items and give it some sort of a name, so that at some point in time we can build a dictionary of sorts. So that, at some point, in time we can go back and define a service request in terms of its components. Then we will go back and define its components perhaps, in terms of their components.

So here I jotted down a simple definition that's says, a Service Request is always made up of the following three items; a customer id number, a machine number, and some set, zero or more iterations of a modification description. Here we have a function partitioning notation, which has come to be known as a data-flow diagram and complemented by some sort of a data dictionary.

Now, after having introduced that notation, I'm going to show you a number of different places, actually four different places, where this notation or this idea, invited by that notation, has come to be seen as significant enough so that someone set his mind to thinking in that pattern and came up with something.

One instance is, the sudden arrival of a brand new kind of computer architecture. An architecture which doesn't have a processor, but has practically inexhaustible resource reservoir of processors, probably small processors, that can be linked together dynamically into any kind of a network. These are called data-flow super-computers.

They break down the essential concepts of both the control-flow paradigm and the control-hierarchy paradigm. That is, the idea that we have only a single stream of consciousness working on our behalf inside a system... because data-flow super-computers have multiple streams of consciousness -- multiple as in truly asynchronous processes in work at any given time.

Here I have just given you two references on the subject of data-flow supercomputers. The first is the current communications of the ACM, an issue given entirely over to Japanese fifth-generation architecture, which is a data-flow architecture. It doesn't have a single processor, it has a configurable array of small processors which are intended to work in legion, in order to accomplish any given thing. We will go back and talk a little bit more about what the insides of a program might look like that would run on a data-flow computer a little later.

But, let me, first of all, talk about a second instance of the data-flow paradigm, coming from a totally different source: the idea that it is convenient to think about systems in terms of moving the data around, rather than in terms of the flow of control. That is a discipline which, unfortunately, was given a grandiose name by a person who will remain nameless; modesty forbids I even mention who he was. In his youth, he gave the name Structured Analysis to this discipline, a very grandiose name, and structured analysis is a discipline for dealing with problems of user-analyst or user-developer communication. A technique which also makes use of data-flow methods. Let me just take those two and go back and show you two very different instances of how they work in detail. Two very different instances of this paradigm, this way of thinking about systems in terms of the perception of the data as it moves through the system.

I will start off with the first one: the data-flow super-computer and mention that a data flow super computer takes advantage of a certain amount of restructuring. Take any old program, and if you read the Dennis article in Computer Magazine for 11/80 on Data Flow Super Computers, he takes a very simple FORTRAN 77 program. He says it happens to have been written using the human paradigm of control flow. It happens to be written as a sequence of N processes, or N components, of the program which are

executed one after the other by a conventional computer. He said it doesn't have to be implemented that way. In fact if you superimpose on top of this the perception of the data, and you ask the questions: what data goes into A and what of it comes out; where does that data go? Does it go directly to B or does some of it go to C, and is it ignored by A? If you look at the locality of all of the data then, in data flow terms, you could restructure that to look like this. There are four processes which are data related in this sense -- which for a supercomputer has some rather interesting connotations because it shows you some of the possibilities for parallelism. A data- flow computer is going to be very inclined to set up not a single process running this, but four processors running it with B and C operating in a high degree of parallelism because they are not strongly data related. The data that passes through B is not strongly related to the data the passes through C.

In Dennis's article, in 11/80 Computer Magazine, the IEEE computer magazine, he cites this as a great advantage. That the computer operates using the data-flow paradigm, but the programmer operates using the control-flow or the control-hierarchy paradigm, and never has to worry about the possibilities or the potential for parallelism.

Well, the second instance of this paradigm is one that says that maybe we should think about it in terms of parallelism. Maybe we ought to think about it in the perception of the data itself. That's what structured analysis is all about, a totally different approach but one that comes up with, remarkably, almost the same answer.

It addresses the problem that you, the builder of systems, talk to a user and come to some conclusion about what the user wants. Then you deliver exactly what you think he wants, and you are rewarded thusly. Because that wasn't what that user wanted at all. What that user wanted is a system that sounded like what he said, but didn't sound at all like what you built. Therefore, we have this user-developer gap, that many of us professional software developers have come to identify as a communications difficulty between the user community and the development community. Well you get the idea from this picture because it's complicated. You see, there it is a uneasy relationship between the user and the snakes-in-the-grass who are approaching her, they are the developers. Or looked at the other way, the poor developer, here all nude and innocent, and the snakes-in-the-grass are the users. A very complicated relationship between those of us who build software and those of them who use the software, who determine, frankly, the final success of the software project.

I came to the conclusion, some of years ago, prior to writing this book, [oh, by the way I didn't mention, modesty I guess forbade, that this is my book]. Prior to writing this book, I was studying analysis methods and communications methods and I was coming to a conclusion, over a 10 year period, starting in the late 60's, that maybe we ought to think about systems, not from the point of view of how the computer perceives them, nor from the point of view of how the user perceives, but from an intellectually neutral point of view. That is how the data perceives the system.

We ought to think about systems in terms of the ways that the data visits the system (this will not look outrageously new to you) and build a kind of a model, a data-flow model. The data-flow model presents the whole system in terms of its subsystems and then presents each one of those subsystems. For instance, number 1 up here, which is the transformation of some 5 inputs into 6 outputs; a lower level partitioning it, again from the data's point of view, to describe the system. Here you will see some of the same inputs and outputs (reservations, lists, equipment coming in, letter for non-return going out), that you saw in the more global view up here, (rental equipment list, notification of returns, letter for non returns, and so forth). I think you saw it at the lower level.

We came to a conclusion that you could model a system in terms of the way the data sees that system. And by the way, when I say system, I'm talking about not just the part inside the computer, but also the part immediately outside--both the automated and the manual portions of the system. We could model it in terms of what the data sees as it passes through the system, by building a data-flow diagram describing the system.

First of all, at this high level abstraction, breaking the system down into lower level pieces, breaking those pieces down into pieces of pieces and the pieces of pieces broken into pieces of pieces of pieces... Each time we have an interface, we give it a convenient name and then we give a data dictionary definition of that interface so that everybody knows exactly what that data is: what the component items of data are that flow in between them. Just as an added level of elegance having this model of the system (having built, in all

practicality, an architectural model of the system (both automated and manual)), in terms of its functions (its transformations of data), data that flows along those component functions, we can now write a specification using that model as a kind of a visual table of contents.

So instead of building one great, huge specification, we write one specification for each micro piece, each incremental piece, and the set of data flow diagrams and a data dictionary that ties those together. This effectively shows how the whole is made up of the sum of pieces. Writing that kind of a specification using this kind of a model, in order to have the user and analyst make a better understanding of what the system is going to be, before they build it, is the technique which I called, in 1975, "Structured Analysis", today I would call it "Analysis by Model Building" or "Specification of Pieces", or some such thing.

Two instances of the data flow paradigm. Let's look at some more.

Recently, we are becoming to believe that the great revolution of the late 1960's and 1970's which brought us block-structured languages might be nearly ended. In fact, I believe as do many of the people that I know in this room that we have come to the end of block- structured languages. The next generation will be what is called an object-oriented language and I can think of two instances of these, Simula and Smalltalk. I didn't give reference on this but, for instance Smalltalk, there was a whole issue of Byte magazine dedicated to object languages in particular Smalltalk, that was August 1981 as I remember.

The idea of an object language, here I focus on Simula because I think it's the prototype of all of them, is a simple one; it says, "When you build a software system, invariably that system, no matter how you think of it, it is a simulation of some other systems". A particular software system is always a simulation of something that happens outside the machine. One example, when you walk in and you do some interaction in your bank there is a automated system which simulates that interaction. So you, a real person, go to a real teller, you exchange a real deposit slip or real check for real money. That teller takes real money out of the cash drawer and puts some sort of an update into the cash drawer to keep a balance so that he or she can quote that work at the end of the day. Meanwhile or slightly later, an automated system goes through all the exact same steps. There is something in there that simulates you, that's an account. There is something in there which simulates the teller, that's a position. There is something in there that simulates a exchange of check for dollars, that's a transaction.

So the idea of object languages is to think of all software systems as simulations, not just think of simulations as simulations, but, think of all software, all applications, as simulations and then build them that way. So when you build this banking system that I've just described to you, if you build it in Simula, you would actually have something , an object, which stood for the person making the transaction. You would have a object which stood for the check and object which stood for a packet of money, an object which stood for the teller, an object which stood for the cash drawer. Each one of these objects is now defined in terms of its stimulus-response characteristic.

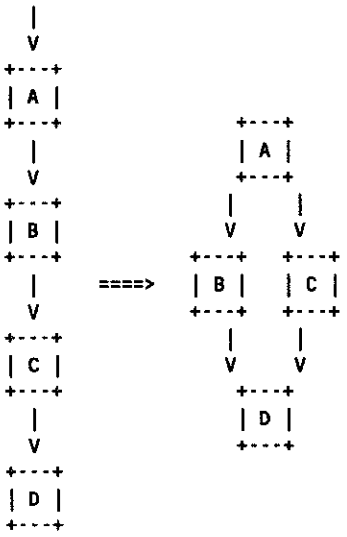
The object that stands for client, when given money, smiles and goes away, or something like that. The object which stands for teller when it receives money, gives back a receipt, updates the cash drawer, things like that.

Now, each one of these objects is allowed to send messages to the other, along a well-defined kind of network and now you can see how this ties in because that network is effectively an instance of the data-flow paradigm. You never actually write the program that decides who does what and when, you simply define each of the objects, you connect them together and you stand back. A program that has no puppet master (well, the operating system, I suppose, which arbitrates the flow of data is, in a sense, a puppet master), but the way you think about it has nothing to do with control, you can never say who's boss -- which program is in charge of which other program. All you can say is that I have created a number of more or less equal entities, objects, a facility for them to send messages to each other, a characteristic of finding each one, and then I get out of your way. In a sense, I implement from the outside and allow them to interact as they work. A third instance of the data-flow paradigm.

The fourth instance, which is a little bit more familiar and considerably more limited, is the idea of UNIX. In particular, a clever idea of UNIX is the pipeline. Many of you are already familiar with pipelines, but if you aren't I encourage you to go out and get the current Byte magazine, October 1983. The defining article was in the Bell Systems Technical Journal for July 1978. The obvious UNIX pipeline is this; it says you, at JCL time, at

command time, can set into motion not just one but several processes. Well, that's nothing new -- we've been able to do that for years.

But, what is a little bit different about UNIX is that you can set those things in motion in a form of a pipeline and that pipeline looks like this:



It looks like 3 asynchronous processes in the data flow notation in which the output of one becomes the input to the next one, and that output of that next one becomes the input to the subsequent one. In UNIX, you request that that structure be set in motion with those 3 asynchronous operating components or tasks, I guess there is only one processor here, but the 3 asynchronously operating components can be set in motion passing data from one to the other along these pipelines. You do that by a simple request say, "Process A, Piped into Process B, Piped into Process C" those 3 are now set up as 3 separate processes, but you look at it as one. You, sitting at the terminal, feed in your net input and the net output comes out to your screen. You are really communicating through 3 asynchronous processes.

Well, if you need more, because B might itself be defined at a lower level to be made up of some of those other components. B1 piped into B2, piped into B3, piped into B4. So you can see, when you ask for the execution of this process, this pseudo process, "A piped into B piped into C", you might really be setting in motion 6 asynchronous processes. What you really asked for is that A start up and pass its output to B1 which passes its output to B2 which passes its output to B3 which passes its output to B4, which passes its output to C which passes its output back to you. So now you can enter some line in and go through all those 6 processes and the process result comes back to you.

Now of course what works at 2 levels works at 3 or 4 so for instance B2 might itself be defined to be made up of B2.1 piped into B2.2 and so forth. What you have, under UNIX, is a simulation of a direct implementation of a data-flow diagram, of the data-flow paradigm. So it says, "Do you think about systems that way?" If you think about systems as I have and negotiation/specification time in terms of the movement of the data, if you have, why don't we build them that way? Just to direct implementation of the data flow model, constructed allocating one process for each level in the data flow model.

Well, unfortunately, UNIX can't do that, direct implementation of a level data-flow model. Unless it begins to get a little bit non-linear. So really you could do a direct implementation under UNIX provided only that all your networks are strings, all your networks are absolutely linear. Making this a fascinating irrelevancy, because of course, you never have a real life function which is a entirely linear network of pipeline processes. So, in fact, UNIX turns out to be tantalizingly close to what you would like to have.

What you would like to have is a kind of UNIX that, if you buy this idea and it's nice to think about systems in terms of data flow, which allowed you to set this kind of structure in motion. And also to be able to say that this is only the top level, that level C really consists of a lower level of 3 or 4 connected subprocesses which transform Z's into F's and G's, whatever those things are, those obviously have to be defined in real terms. But, if you could do that, you would have a two-dimensional UNIX, if you will.

I will lay down why you can't have that. Reason 1 is that UNIX has a job control language structure, a command structure. Language is intrinsically one-dimensional. You see, as long as you are required to convey the network form by giving 1,000 words, that's equivalent to the picture, you're really going to be limited to linear networks. Because language is intrinsically linear.

There's been a lot of talk recently about the idea of a non-procedural language. I submit to you that the idea of non-procedural language is a total contradiction in terms, because of the way the human brain is structured. We've got two parts of the human brain; one part which is intrinsically procedural, that's the left side of the brain that

deals with language. The other part is intrinsically non- procedural, the part that's used for pictures, that deals with holistic concepts, that deals with relationships, that's the right side of the brain. Therefore, a non-procedural language is, in a way, a contradiction in terms.

What you would like to have, that could honestly deliver on the bankrupt policy, the bankrupt promise, of the non-procedural language is a non-procedural non-language. By that I mean a non-procedural approach that involves entirely deserting the language approach and going directly to pictures. Instead of having a representation in language of this network, we will now draw the network.

So reason number 1, why you can't presently have the two-dimensional UNIX is that up until now we haven't had the possibility of dialoguing very nicely in pictures with a computer. Of course, there are a lot of computers that put out pictures, but invariably they put out pictures after you type in 1000 words that are the equivalent of pictures. What we haven't had until very recently is a computer that is easy to put picture requests into.

If you have such a mechanism, then you could not tell the computer what you wanted to do, but you could show it what you wanted it to do by drawing that network in view of a job control request. You would be drawing, not writing, a job control picture. One of the reasons that we haven't been able to do this so far is the lack of good facilities for graphic input.

Another one of the reasons is that our languages have really not been happy about the idea of asynchronousing. Oh, sure you print something like the 'Attach'. That is a donkey trying to do a horses work. It is just something which is difficult to think about, difficult to work with, difficult to develop. It was not really intended as part of the original Pascal language, it was not thought out, it was kind of an add-on.

So one problem is language and one problem is the idea of the job control picture -- the graphics.

The closest thing we've got has been the simulation languages which solved one of the two problems in that they did allow you to nicely and elegantly have the ability to set up this network of asynchronous processes, each one simulating something. This is a language developed by Aurel Soceneantu at Brigham Young University called Sencro and the language is a simulation language where each one of these is some sort of a simulated activity, then these are conditions. One of the principle conditions is flow of data. Thus back to the data-flow paradigm.

This introduced two more problems, the first is that while it can implement that structure there is no nice way to tell it to implement that structure except by typing down in network connection terms connect if you want to receive word before it connects P1 to C4 to P4 and connects P1 to C3 to P3 which could drive anybody bananas. We can't show it what to do, we can only tell it what to do.

In addition, it introduced a second problem which is: this is a language, the whole thing is a language. That means that in addition to building the ability to asynchronise all of these components, you also have to build the ability to do what each one of the components does. By the way, Smalltalk works exactly this way. It's an elegant way to set a structure of asynchronous objects in motion communicating with each other, but God save you when you have to write the definition of one of those objects. The language is very nice for showing the interaction of the object, but is a disaster for describing the objects themselves.

Therefore, starting in 1972, working with members of the Systems Guild, we started to investigate the following split. We said, "let us allocate to the operating system or the shell of the operating system the business of showing how the various components communicate, what data flows among them, and to the language, the definitions of the insides of those processes." So instead of having a language that does both, we allocate the interaction to build these and, setting in motion of the network, we allocate that function to the command shell and we allocate that function of the inside to a language.

Well, now we can use any of the number of languages. Let me first of all show what machine we put this on. We put it on any of this new generating or half generation of hardware that started to come along. One of these devices that's got huge processor, typically a bit-sliced processor, like Lilith, Star, Apollo, or even Lisa, with your primary fat lines here to show a very wide front-end interface between the processor and a high resolution bit-mapped video screen, which allows you to put out elegant graphics, and a cursor mouse that allows you to put in quality graphics, and a printer that allows you to print those graphics, and a network that connects you together so that you can pass these things, both graphic and text, back and forth.

This is the kind of a processor that you are seeing today in many different places. Here, I'll show you just a screen taken off of the Lilith processor. It is a window-managed screen that allows us to deal with... the screen is filled with a set of windows and each one of these could be used to put something different in, each one has a separate scroll bar so you could work the cursor over here with the mouse and scroll this list thing or you can scroll through the data then you can pop this window onto the top. I'm sure you have seen window-managed screens... that's just the side effect of what you get when you have one of these machines, you also got a very nice possibility for graphics. So here we have some graphics. It shows how a bit mapped image in memory can be transmitted to the screen. Since I'm interested in the whole idea of data-flow modeling, let me show you a data-flow model.

What I want to talk to you about is something that allows you to draw these data-flow diagrams, then does them for you. So not just a graphics system, but an execution system.

So now we get the hardware, typically the cursor mouse has the ability to put pictures in as elegantly as we can put pictures out. What we need now is a language that supports that, and it turns out that half a generation of languages afoot, 3 of which are Ada, Mesa, and Modula-2. Ada from the United States Department of Defense. Mesa is a yet-unreleased product, probably never to be released product from Palo Alto Research Center...

There's a kind of interesting story about the development of not Ada, but a language that was until the 1960's what Ada is perhaps in the 1980's and that is Algol 68. Niklaus Wirth was actually part of the committee, and the enormous deliberating body whose charter was to develop the ultimate block-structured language to be called Algol 68. The modus operandi which was that he was to get everybody in a room like this to lay their wishes on the table and implement the union of all wishes.

?, Niklaus Wirth and a number of others were part of a very famous ? that walked out one emotional afternoon, and Wirth is a person who is not given to outbursts, he made the following mistake whether out of emotion or not I don't know, he seems like a very calm man to me. He said you people are profoundly misdirected you are trying to design a language by committee. A committee with a hundred people in a room sitting around a great huge table, he said you are profoundly misdirected you can't design languages by committees.

Wirth said only one person, one mind can build a harmonious language and I will build it. I will go on and I will develop the language the one that you should be developing. I will come up with the things that you wouldn't be able too. I will code it, I will debug it, I will deliver it, I will make free tapes available for anybody who sends me a piece of software and document it elegantly and I will be finished before this committee can meet again. Thus was formed Pascal. What ever happened to Algol 68?

Comes the late 1970's, the language was born in ? mind called Ada. How did he develop this language? Well two fundamental rules in developing this language; 1 is that it ought to be a Pascal based language, because everybody knows Pascal is harmonious and beautiful because it's got the thinking of one very clear thinking individual, that's rule number 1 it will be Pascal based.

Rule 2, is we will conduct a huge committee, get all our ideas out on a table and we will implement all the ideas. They asked Wirth to participate in this, and he did in fact participate in Ada for a second or two, at which point he said hey, you guys I mean you are headed in profoundly the wrong direction. You just can't build languages by a huge committee and throwing a list on the table, only a singly harmonious mind can build a language, I will go and I will build the language that you people should be building and I will specify it (that they had so much trouble doing), I will specify it and I will code it and I will debug it and I will deliver it with a distribution tape and fairly elegant documentation and I will have it done before this huge committee can meet again. Thus was born Modula-2. Born between the ? specification and the ? specification.

So if history repeats itself, I say Modula-2 is to Ada as Pascal was to Algol 68. Perhaps that's true. In any event I can say Modula-2 is up and running, is workable today it is implemented on dozen of different machines - on UNIX on Lilith machine on the Apollo on any number of 8 bit machines and 16 bit machines on the Sage on the IBM PC - through many different organizations ? Particularly, Volition Systems a very elegant implementation of Modula-2, p-machine running on the Sage processor among others.

Among those languages I think the one that is most interesting to me for my purposes was the

Modula-2 language. So we set out working with some students at Brigham Young University, students who are fairly star and willing to work for peanuts, helped us to construct a new kind of a (I won't say a operating system) a command processor one which allows you to implement directly this structure. A kind of two- dimensional UNIX.

We set to work to build a command structure in which components themselves, coded in Modula-2 a language which supports elegant asynchronism, an operating system which allows us to set 3 or n of these at work passing data along named pipelines, data of known composition all of them operating together that allows us to set the structure in motion not by telling it what to do but showing it what to do.

So, you sit down at your screen you've go a high resolution bit mapped video screen like a Lilith, Diser, Lisa or Apollo and working with the mouse you begin to draw your requests. The requests are in the form of a network; you put ? connections among components of the network. The elements of the network are themselves Modula-2 processes.

Modula-2 is nice for two reasons, all these forth generation or third and a half generation languages are nice for two reasons. One is they support ? synchronism and the second reason is the idea of encapsulation. You can provide a set of highly concealed facilities for intercommunication, you can export that to alternate producers. I export to you, you write a process for me, ? to ready process ?. I export a set of facilities to you, they are known to you in kinds.

There is one of two modules, the definition module that says what the contract is between the two of us. The contract is there are five procedures: ? name pipe, ? name pipe, ? name pipe and so forth. You use those procedures, you restrict the conviction, you pass the parameters exactly as it says in the contract module, and I who have built the semiprivate implementation module will make sure that your hearts desires are carried out. I export that ? you imported for me and the actual details on the ?.

So the language has good asynchronism and a good possibility for sharing of communications facilities that ought not be make public to everyone. Nobody has any right to know what Pipeline is. I've defined a type of Pipeline I export that to you. It's an opaque type, that means I won't say anything about it. You can define something to be of type Pipeline, you can name it but you can't tinker with it. Frankly just between us, whether it's got an integer and a pointer and a ?, that's none of your business. You can operate a product using these imported facilities from me. This allows you to write something which interacts with me and Pipelines.

Once you have written that process, and we have written a whole bunch of them, we've built ourselves a library of these. Then I can sit down at my terminal at my workstation and I can begin to draw a job control request. I can call up a menu which allows me, selecting with the cursor, to place anywhere on the screen processes or icons representing devices and files or literals or connections between those things.

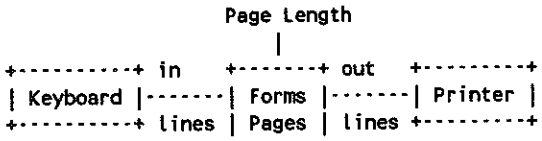
So in particular, what I can do is I can set up this procedure I can set that procedure in motion by pulling it and saying go. (I'm going to come back and talk to you about the details of that procedure). Just in general what that procedure is, is 5 asynchronous tasks connected to the other through named Pipelines; the input seems to come from the keyboard and the output seems to go to a printer. I have made a job control picture, if you will, and asked for that to be done. I have a good drafting facility to draw it, but more important I've also got a command shell that starts that in motion that starts up those 5 asynchronous tasks and passes the data among them. By the way none of the activities none of the tasks knows anything about the topology of the network.

Notice that each connection, unlike the convention that I introduced earlier where each data flow has got a single name which refers to the data that goes over it. In the implementation each connection has got two names, it's the connection of two ports. One of the ports is the name as known inside here and the other one is the name known inside here. Who knows that the output that this procedure calls copy2 is connected to the input that this one calls ? Because I made that connection with my little cursor mouse. I'm the only one that knows what the topology of the network is; well I am now being command processor that actually starts that moving.

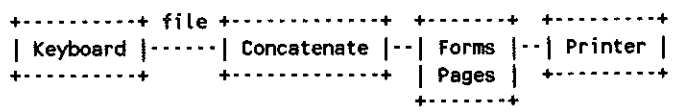
Let me build that process for you little by little. Imagine yourself sitting at the screen and using this menu to select icons and now you are going to select and connect three of them to build a very simple text processor. I'm going to a example of a text processor, because I don't have to explain to you what text processor does. I mean you know

you type in text and it come out in some printable format. I don't have to explain what that process is all about. I am going to show you a very simple text processor built up little by little.

The first version of this makes use of only one process it's called form pages. When this process is on all it does is read in lines and spit out lines, only it knows its talking to a printer or some such thing that leaves some blank space at the top and the bottom so you don't print over the perforation that's it nothing else.



I connect a keyboard to its input port, I connect the printer to its output port, I neglect to connect this third port which means it takes a default. A default is what ever the default the printer is defined to be. It doesn't know there is a printer out there put it has a built in default. I could also connect the page lengths to the keyboard and it would ? the page length. I could connect it to a file that has the page length in there. Another process that knows the page length or I could even put a literal in there. As you cans see a very simple a very dumb wordprocessor. I sit at my keyboard, many thousands of dollars worth of hardware tied up in this, and enter line after line after line and they all come out on the printer except it never prints over the perforation, that's all it does for me besides that the output is identical to the input.



I could now add another process into this network, and the other process I am going to add in is one that concatenates. Now instead of typing in text I type in file names, it finds those files concatenates chapter 1 with chapter 2 with chapter 3, if those are the names I put in, forms a huge document puts it out a line at time into formed pages which allows it to be out on the printer. Now that's a slightly more elegant textprocessor, but I'm not done yet.

I'm going to construct something a little bit more elegant. I'm going to add a little value into this, to say my limitation length is not 66 but 35. Now that's very nice if I reconnect this so instead of going to the printer it goes to a screen which has 35 lines on it. Now in my job control request I actually specify a literal value. Well that's good, it puts out a 400 page document in pages very quickly though, so you can't read it to well. So what I'm going to do next is add some sort of a process which puts on hold the output until I say, "Go ahead I've read this one." So now I have exactly the same two processes and I've added the display page at a time. You could write the code for that in your head, I mean while you are sitting here. You know what it does it reads in a line, puts it on the screen, then it waits on the keyboard for a character to let the thing go ahead.

I'm going to go back to the printer and again add some of the features that we all know and love in text processors. Still notice the concatenation, pages are still the same but now I've got one that fills it also needs to know the line length but I've left that to the default in this particular case.

When you add filling you might as well add adjusting and I guess that brings me over to here. So here I've got the filling and adjusting and I want the same line length to go to both but I only want to enter it once, so I use the copy process. Now I've got five processes which together cause it to asynchronous implementation with a very simple text processor.

You could implement the simple text processor with a hierarchy, there is no question about that. But there are some very nice reason why this is so much better. One of the reasons, if any of you are familiar with what Michael Jackson calls a structure clash, there is a terrible structure clash in the text processor and it is rather neatly right here. I won't go into that.

Since I'm running a little short on time, let me just say that each one of these processes could be a Modula-2 program or a Modula-2 process which imports ? one facilities. Each one could be but in fact some of them are not. Some of them are defined as low level networks. So take that facility called fill which does text filling which realigns the lines so that they are to the left, is in its self defined as a lower level network which splits the text into words and then recomposes the words into filled lines.

Alternately, it could be defined a little bit more elegantly using the same splitter and more or less the same filler with a hyphenator to create a filling process which does hyphenation as well.

Then again I think you could see what the code at the very bottom level would look like for each one of these.

Let me add one last piece of information and that is a reference for you to read about what I think is the worlds first two-dimensional operating system or two-dimensional command shell, it's a paper written by myself and ? called "Synchro: A Dataflow Command Shell for the Lilith/Modula Project". Now I'm being a little optimistic in telling you when it is going to be published because it has only been tentatively accepted; I expect it to be published in the Orlando Conference, The 7th International Conference on Software Engineering. If it's not published there I'll make sure it's published somewhere else.

Efficient p-System Code

By: **David Gelfand, Ferox Microsystems, Inc.**

From the October 1983 USUS Meeting

There are a number of ways to measure efficiency: compact code, execution speed, memory usage, program structure, disk usage and a lot of others as well. I am going to address the issue of compact code; how do you write a program that is as compact as possible and still performs the job it is supposed to perform?

The benefit of compact code is less disk space required for the program. A lot of times less disk space is not important because you're not developing big programs -- until the program grows and eventually you run out of disk space or you want more space on the disk for files. So you should not wait until you need the room before you start worrying about efficiency. You should try to adopt efficient practices so that as you develop your program it is initially more efficient.

Other benefits of compact programs are that during the execution less disk swapping occurs. The p-System is going to swap code segments in and out as it needs to, and if these segments are more compact there will be less need to swap. In some cases writing compact programs results in faster execution.

The cost of compact code may be to sacrifice efficiency in other areas.

The first thing that you have to address is how to measure alternative methods of doing things to determine which of the alternatives is more efficient. One possibility would be to learn how your compiler works and take a look at what p-codes are going to be generated by whatever statements you are using in your program. You could measure alternative methods that way, but that is rather difficult.

A simpler method is to write a test program. Within that test program declare two segment procedures, each using an alternative method to accomplish the some thing. Compile the program and use the library program to look at how big each one of those segments are and then compare the two methods to see which is more efficient.

Just to show you how I did it -- I wanted to compare the use of "+1" with with the successor function ("SUCC").

```
PROGRAM test_add;

  SEGMENT PROCEDURE use_add;
  VAR x : INTEGER;
  BEGIN
    x := x + 1;
  END;

  SEGMENT PROCEDURE use_succ;
  VAR x : INTEGER;
  BEGIN
    x := succ(x);
  END;

BEGIN
END.
```

I wrote a program called test_add that has two segments. In one I used addition and in the other I used SUCC. That's all the program has to do; you then execute the program and look at its components.
Using LIBRARY.CODE:

1. Execute "LIBRARY".
2. Enter "DUMMY" for the output file.
3. Enter the name of your compiled test program for t
4. The components of your test program will appear o the screen.
5. Type "A" to abort.

Here's how to use the library. First you execute program LIBRARY. It does not matter what you name the output file because you're not actually going to be outputting to the file. Then enter the name of your compiled test program for

the input file; the components of your test program will appear on the upper half of the screen. For example:

```
0 p TESTADD   18
1 s USEADD    20
2 s USESUCC   20
```

The first one is the program itself, there are 18 words of code associated with program testadd. In this case the two segment procedures, useadd and usesucc, came out to be 20 words so neither method is more efficient. You should then abort from the library.

Case statments. The first thing I look at are CASE statements. When you use a CASE statement in your program the compiler is going to build a table in your code file and it's going to allocated one word for every element between the lowest and highest value that you mentioned in your CASE statement. So if you had a CASE statement that has 1, 2 and 3 as its cases it is going to build a table of three words.

The advantage of a CASE statement is that it has faster execution than an IF statement. If you avoid using case statements you are going to be sacrificing some execution time.

Below are three different procedures that accomplish the same thing, using CASE and/or IF.

```
SEGMENT PROCEDURES case_1 ( in_ch : CHAR );
{76 Words}
BEGIN
  CASE in_ch OF
    'Y', 'y' : writeln;
    'N', 'n' : writeln;
  END;
END;

SEGMENT PROCEDURE case_2 ( in_ch : CHAR );
{60 Words}
BEGIN
  IF in_ch IN ['a'..'z'] THEN
    in_ch := chr ( ord(in_ch) - 32 );
  CASE in_ch OF
    'Y' : writeln;
    'N' : writeln;
  END;
END;

SEGMENT PROCEDURE case_3 ( in_ch : CHAR );
{51 Words}
BEGIN
  IF in_ch IN ['Y', 'y'] THEN
    writeln
  ELSE
    IF in_ch IN ['N', 'n'] THEN
      writeln;
END;
```

In case_1 there are two possible cases, 'Y', 'y' and 'N', 'n'. This is going make a table from 'N' through to the 'y'. So it will allocate space for all possible characters between 'N' to 'y'. One way to make that more efficient would be to capitalize the letter before using it in the CASE statement - that is what the second procedure does. It checks to see if it is lower case; in that case it subtracts 32 from it. In case_3 I used an IF statement. Here, it turned out the IF statement was more efficient. That's going to depend on how wide a range there is between the lowest and the highest value in the CASE statement.

Other examples of CASE statements:

```
SEGMENT PROCEDURE case_4 ( in_num : INTEGER );
{1041 Words}
BEGIN
  CASE in_num OF
    0 : writeln;
    1 : writeln;
    2 : writeln;
    999 : writeln;
  END;
END;

SEGEMENT PROCEDURE case_5 ( in_num : INTEGER );
{47 Words}
BEGIN
  IF in_num = 999 THEN
    writeln
  ELSE
    CASE in_num OF
      0 : writeln;
      1 : writeln;
      2 : writeln;
    END;
END;
```

Case_4 is very bad programming practice because 1041 words are created in the code file. In Case_5 all I did was make an exception for the 999 and then jump to the CASE statement.

Range checking. When you leave range checking on during compile, extra bytes of code are generated every time you address an array or sub-range. Those extra bytes are going to add space to your code file and also slow the program down.

The benefit of range checking is that it helps you to trap errors during the development of the program, but turning it off results in faster execution. For example:

```
SEGMENT PROCEDURE range_on:
```

```
{23 Words}
VAR dummy_array : ARRAY [0..2] OF INTEGER;
{$R+}
BEGIN
  dummy_array[1] := 100;
END;

SEGEMENT PROCEDURE range_off;
{22 Words}
VAR dummy_array : ARRAY[0..2] OF INTEGER;
{$R-}
BEGIN
  dummy_array[1] := 100;
END;
```

The first procedure turns range checking on, "{$R+}" which is the default value for the range checking compiler option. If you want to turn it off, use the second procedure with "{$R-}".

Low-level intrinsics. Another thing to look for is the opportunity to use some of the low-level intrinsics. A lot of people become familiar with these when they have been working with the system for a while. They are very efficient and very fast; their execution is much quicker than the way you would generally program it. The danger is that no range checking is performed using these operations.

FILLCHAR will fill an array with a given value, MOVELEFT and MOVERIGHT are used to move a string of values from one array to another. Some examples are below.

```
SEGMENT PROCEDURE init_ala_loop;
{31 Words}
VAR init_array : ARRAY [0..10] OF INTEGER;
    i          : INTEGER;
BEGIN
  FOR i := 0 TO 10 DO
    init_array [i] := 0;
END;

SEGMENT PROCEDURE init_ala_FILLCHAR;
{21 Words}
VAR init_array : ARRAY [0..10] OF INTEGER;
    i          : INTEGER;
BEGIN
  fillchar (init_array, sizeof (init_array), 0);
END;
```

The first procedure initializes an array of integers to zero with a FOR loop. The FOR loop requires 31 words of code. The second init_ala_FILLCHAR; gave it the array name, the number of bytes that you want to fill, and the value of the character that you want it filled with.

Examples of moving:

```
SEGMENT PROCEDURE move_ala_loop;
{36 Words}
VAR source : PACKED ARRAY [0..20] OF CHAR;
    dest   : PACKED ARRAY [0..10] OF CHAR;
    i      : INTEGER;
BEGIN
  FOR i := 0 to 10 DO
    dest[i] := source[i];
END;

SEGMENT PROCEDURE move_ala_MOVELEFT;
{22 Words}
VAR source : PACKED ARRAY [0..20] OF CHAR;
    dest   : PACKED ARRAY [0..10] OF CHAR;
    i      : INTEGER;
BEGIN
    moveleft (source, dest, 11);
END;
```

Move-ala-loop just moves values from one array to another using MOVELEFT. Again, the first example is a FOR loop, I am moving some elements from source to destination. The FOR loop required is 36 words and MOVELEFT required 22 words and will execute much faster; consider using these procedures.

Variable ordering. The order that the variables are declared can effect the efficency of your code file. The first 32 bytes of variables declared in a given block can be addressed with a short load p-code. What should be done in the program is try to determine which of the variables are most often used throughout the procedures and declare those in the first 32 bytes of the declaration. This will make the code file smaller and also result in faster execution.

```
SEGMENT PROCEDURE misplaced;
{25 Words}
VAR dummy_str : STRING;
    x, y, z : INTEGER;
BEGIN
  x := 1;
  y := 2;
  z := x + y;
END;

SEGMENT PROCEDURE well_placed;
{22 Words}
VAR x, y, z : INTEGER;
    dummy_str : STRING;
BEGIN
  x := 1;
  y := 2;
  z := x + y;
END;
```

In misplaced four variables are declared; a dummy string, x, y, and z. Dummy_str is going to require 82 bytes so that is going to push it beyond 32 bytes right there. By just reversing the order, as in well_placed, making the x, y, and z declared in the first 32 bytes cuts off 3 words.

String constants. One misconception that people have is that string constants can be used to cut down on code space if you are going to be writing them out over and over again. This is not true, the compiler is going to recreate the string in the code file every place it is used.

```
SEGMENT PROCEDURE no_const_write;
{66 Words}
BEGIN
  writeln( 'TEST STRING' );
  writeln( 'TEST STRING' );
  writeln( 'TEST STRING' );
END;

SEGMENT PROCEDURE const_write;
{66 Words}
CONST test_string = 'TEST STRING';
BEGIN
  writeln( test_string );
  writeln( test_string );
  writeln( test_string );
END;
```

No_const_write and const_write compile out to the same number of words. My recommendation here would be to declare a dedicated procedure if you are going to give a message out many times throughout a program.

Sets. Another possible tradeoff is using a SET versus using an IF statement. the tradeoff is going to be determined by how many possibilities there are. in the example below there are only two. If there were ten possibilities the IF statement would be less efficient.

```
SEGEMENT PROCEDURE use_SET;
{35 Words}
VAR ch : CHAR;
BEGIN
  IF (ch IN ['Y', 'y']) THEN
    writeln;
END;

SEGMENT PROCEDURE use_OR;
{27 Words}
VAR ch : CHAR;
BEGIN
  IF (ch = 'Y') OR (ch = 'y') THEN
    writeln;
END;
```

External procedures. Another common mistake I see occurs when people develop a program that uses external procedures. When two different units are going to use the same procedure many people will compile the units separately and LINK in the external procedure separately to each unit. when they add that unit to their program that external procedure is going to appear twice. My next example illustrates this.

```
UNIT unit_1;
INTERFACE
  PROCEDURE unit_1_stuff;
IMPLEMENTATION
  PROCEDURE osm_stuff; EXTERNAL;
  PROCEURE unit_1_stuff;
  BEGIN {unit_1_stuff}
    .
    osm_stuff;
    .
  END;
BEGIN {unit_1}
END.

UNIT unit_2;
INTERFACE
  PROCEDURE unit_2_stuff;
IMPLEMENTATION
  PROCEDURE osm_stuff; EXTERNAL;
  PROCEURE unit_2_stuff;
  BEGIN {unit_2_stuff}
    .
    osm_stuff;
    .
  END;
BEGIN {unit_2}
END.
```

These two units use the same external procedure osm_stuff. In this case you should use a middleman unit such as in the example below.

```
UNIT middle_man
INTERFACE
  PROCEDURE do_osm_stuff;
IMPLEMENTATION
  PROCEDURE osm_stuff; EXTERNAL;
  PROCEDURE do_osm_stuff;
  BEGIN {do_osm_stuff}
    asm_stuff;
  END;
BEGIN {middle_man}
END.
```

This middle_man unit gives all other units access to that procedure. This unit might be nothing else but call that external procedure. By making this middle_man procedure you can

compile once, link once and then your other units will use the middle_man unit, so that the code for the external procedures only appears once, as follows:

```
UNIT unit_1;
INTERFACE
  PROCEDURE unit_1_stuff;
IMPLEMENTATION
  USES middle_man;
  PROCEURE unit_1_stuff;
  BEGIN {unit_1_stuff}
    .
    do_osm_stuff;
    .
  END;
BEGIN {unit_1}
END.

UNIT unit_2;
INTERFACE
  PROCEDURE unit_2_stuff;
IMPLEMENTATION
  USES middle_man;
  PROCEURE unit_2_stuff;
  BEGIN {unit_2_stuff}
    .
    do_osm_stuff;
    .
  END;
BEGIN {unit_2}
END.
```

Some miscellaneous items. Unnecessary use of segment procedures. Every time that you create a segment procedure the compiler is going to want to put it at the beginning of a block. Lots of segment procedures are going to result in a lot of filling of the end of the previous block, a lot of unnecessary code space. Also, additional overhead code is going to be created for the segment procedure.

Also, unnecessary use of native code should be avoided. This is because native code increases the size of the code file and it doesn't always result in faster execution.

Finally, avoid unnecessary use of big units to do little tasks. A lot of times you will buy a p-System that will have some great units in it that do all sorts of things with your machine, but it might not be necessary to use those units to accomplish that. What you are going to do is add that entire unit just to get at that one procedure. A misconception is that the selective USES statement will take care of that situation, but it is not true, a selective USES statement will only affect the identifiers called in during compilation; during execution the entire unit is still there.

In summary, in order to have compact code you should: check your use of CASE statements, turn off range checking, don't SEGMENT unnecessarily, don't use unnecessary native code, only link in external procedures once, use low level intrinsics when possible, consider the order of variable declarations, selective USES won't help, and consider your use of others' units.

Guerilla Guide Update

By: **Bart Thomas**

DAMAGE CONTROL

There are a number of things that can go wrong with the p-System, just as with any operating system. While you may insulate your software from most of these, it can (and will) happen sooner or later, whether due to disk problems, power glitches, or whatever. When these do happen, keep certain things in mind.

1. Keep your Cool! Perhaps, even to the extent of, after following rule 2, putting the whole project aside, and doing something else, while you run your options through your mind.

2. Before you do anything else, make a clean copy of the disk where you had the problem (say, a lost file). Then do everything else on the copies (I find the more copies I make, the easier the problem fixes... Probably because I have time to think while making the copies!)

3. If your file has become lost, it may only be lost to the Disk Directory. Before you start playing around with disk editors, try some options in the Filer!

4. Even if you wind up with two files of the same name, (Yeah, I thought that couldn't happen either!), You can use the Filer to help you out!

This all happened to me once, using Quickfile, which depends upon a file called QF.FC to do its own file handling. Apparrently, this file is open while you are running the program. At any rate, for reasons I do not understand, I was saving an updated file, when the message "Cannot read File disk" came up on the screen. After trying everything I could think of from within the program, I quit and loaded up Pascal to see what

had happened.

I compared the disk I had with other Quickfile File disks. (I had not been backing up as much as I should have!) The disk was missing the QF.FC file and the DataFile QFF.I was shown twice in the disk Directory. All this meant was that I was faced with the potential of having to rekey a mailing list of some 200 clients! Then I noticed that there was a "hole" of 2 blocks in one group of files... Just the length of QF.FC! On a COPY, I went into the filer, and using M)ake, Made a file, QF.FC on that spot (this is well documented in the OpSys Manual), and tried it. It worked!

That left me with the double file on the disk. Since I had had the problem while saving a file which had had only a minor change, I was not too concerned with which version I deleted. (QF does not use a Filer type dating system!) Using the Filer again, and COPY disks, I transferred the files to another disk, using the T)ransfer option and File Name. Whenever I gave an Instruction for one file, it applied to both, so I wanted to get them onto another disk. As the first one transferred, all went well. Then as the second got ready to go, I was (of course) asked if I wanted to delete the old version (on the destination disk). All I had to do was hit 'Y' and the file was by itself!

Later, experimenting, with a suggestion from Jim Harvison, I found that I could just as well have used the R)emove command, with the file name followed by a '?'. Answering 'Y' to one prompt, and 'N' to the other, worked just as well.

PRINTING.

This is a lot better than the solution I showed in the first version of this guide. In a Program, you use the following commands: (Thank Arley Dealey for this elegant approach!)

```
PROGRAM Xyzzy;
  Var: Printer : INTERACTIVE ;
  BEGIN { Xyzzy }
    RESET ( Printer, 'printer:' ) ;
    WRITELN( OUTPUT, 'This prints on the console.') ;
    WRITELN( Printer, 'This prints on the printer.' ) ;
    CLOSE( Printer )
  END { Xyzzy } .
```

Arley points out that closing OUTPUT is questionable form. (You could get locked out of using the console!) He also points out that Interactive is a UCSD extension, so if you are taking an ISO Pascal Course and doing homework on the Apple, you could substitute TEXT. INPUT and OUTPUT are predeclared in Apple Pascal itself and must not be redeclared by the user.

Note also that the second parameter to RESET is a string. Therefore, the statement must be in the form above (with the string included in (Pascal) (i.e, single) quotes, as above.

FILER STUFF.

The first Filer watchout is be careful when copying disks, etc. If you get the message "warning vols 4: and 5: have the same name" (or words to that effect), before you do ANYTHING else, if you have not changed disks, you should take steps to avoid having the system confuse which disk is which!

In the words of Randy Bush (USUS News vol 9/10 p.45), you have just left the real world!!

You MUST NOT bluff it through by using only volume numbers, as in, following a T)ransfer command, #4:zzzyyy.text,#5:$. Sooner or later, you will have a disaster doing this!

Rename one of the disks, using the C)hange option. Once you are done, you can remove the original and then rename the destination, by again using the Change option. While this is being done, it is a good idea to have the door to the drive of the disk you do not want changed opened, to prevent an overwrite!

If you just want to see a textfile, it is not necessary to get it into the Editor. From the Filer, you can transfer it to #1: , which is the Console and see it on the screen. Control-S will toggle scrolling on and off so you will have time to read it.

If you are transferring a file under the same name, you can use the $ sign to move both text and code files. Thus, if you have files Bopper.text, and Bopper.Code, you can (in response to the Transfer? Prompt) type: Bopper=,#5:$ to transfer both files.

DISK PROBLEMS!!

While we are dealing with the FILER, an *ALERT!!* If you start getting messages to the effect of volume went off-line, or a V)olumes command from in the FILER does not show the disk volume you have in a drive, you may have formatting problems.

These are insidious. As most of us keep a supply of formatted disks on hand, consider this scenario: As a careful owner, you take your drives in for a check-up. The techie fixes them up and reports that he corrected an alignment problem, or recalibrated, etc. You *should* ask him if that will have affected disks formatted by that drive! In my

case, it was the Carriage Limiter.

If he says it will, mark *all* your blank formatted disks as UNFORMATTED !! I did not do this, and started getting the sort of errors I mentioned. Another source of this sort of error is the centering of the disks, so into the dealer I went to have the drives checked! You guessed it! The problem was that the disks had been formatted when the drive was bad!

MORE LANGUAGE!

Another quickie... I've noticed that very few working programmers bother with the distinction between normal ends of lines and those preceding 'End' . The compiler will accept the line ending ';' , and having it there will avoid having to stop in the middle of the compile to insert it because you forgot to when you added more lines to the program.

Once you've started to get a feel for how the language works, you will find it worthwhile to browse around in LM. among other things you will note that anything in printed commands in that manual or in file names that appears in the in the square [..] brackets, may legally be omitted.

EOF and EOLN.

Beware of the different ways in which EOLN is handled in text vs interactive (keyboard vs. disk) files. It can screw up your error checking routines. (LM 30, 34) What happens is that EOLN from the keyboard is read as a space (' '). If what you want to do is check for EOLN, you must specifically do that! This can be accomplished with code such as this:

```
Var CH : Char;  { In both samples }

Read (CH);
If (CH = ' ') AND (EOLN) then
  {Whatever you plan to do at EOLN }

***  OR  ***

Read (Keyboard, CH) ;
IF EOLN ( Keyboard )  THEN
  {rest of code...}
```

Although I have not tried it, this could also precede a CASE statement to handle other responses.

```
IF ( EOLN ) then
  Exittest := True;
  Else
   Case (CH) of
     'c' := etc.
```

DISK FILE I/O.

Note that it is also possible to set up a 'menu' to select which files are used. Then you can simply use a CASE structure to select the filename as below.

```
Case F of
  1: Inname := 'DISK:File1.text';
  2: Inname := 'DISK:File2.text';
End { Case } ;
```

One hazard is that you must not leave a cosmetic space between the Disk volume name DISK: and the filename, as the space will be interpreted as part of the name.

THE COMPILER.

If you get *any* error message in the compiler, a code file will not be generated! Thus, you cannot expect to have a code file, even if there has been a last minute message before you exited, unless the compile has been clean! (This is particularly painful (and sneaky) in the case of unsatisfied forward references!!)

If you are compiling a longish file, and wish to generate a List file (compiler option L+). This needs a LOT of room on the root volume. You should use the technique shown in the //e file to make a disk to use in place of the normal root volume. For that use, you only need the System.Pascal and System.Library files on the substitute root volume, but the Syntax and Charset files are handy and do not take up that much room. For even longer files, you may find it simpler to send the List to the Printer using the L Printer: option. After all, you are going to print it sooner or later anyway, and this is much simpler than trying to squeeze those last few blocks onto a crowded disk.

PARAMETERS.

While the subject of parameters is covered fully in most texts, there is a little gotcha associated with them, so let's review for a moment.

A variable parameter is used when we want to be able to change the value of a variable. A value parameter is used when we do not want the value of the variable to be changed by the procedure we are passing it to. We distinguish between these by preceding variable parameters with VAR in the header of the procedure in question. So far, so good.

In our dedication to protecting our variables, however is a pitfall! Remember that a procedure

can alter the value of a value parameter, but cannot return the changed value for use outside itself! Now suppose one of your variables is a file of addresses which uses about 5k of memory, and you dutifully pass it as a value parameter. Since its value *can* be changed for use within the procedure, Pascal protects the original value by reserving enough space for the whole file in the beginning of the procedure. (If you use the L option in the compiler, you'll see that 5k being used!) If this is done enough times, your program will crash with a Stack overflow!

The answer, which appears in some of the texts is to pass your files, if they are held in memory, as Variable parameters. If you don't change them, they'll stay intact! Besides, now your program will run!

STILL MORE LANGUAGE:

Another potential gotcha lies in the handling of Integer variables. We all know that if you mix an integer and a real in a calculation, the result is a real, right ? Well, it would seem that if you were to do this in the process of assigning a value to a real variable, there should be no problems, but, there is a problem here, too!

In a long program, dealing with interest rates, I had the following variables:

```
Var Face, Tsc : Integer;
    Value, Cpn, : Real;

Cpn := 12.0;
Face := ????  < This is the Joker >
Tsc  := 60;
Value := (Face * 1000) * (1-(Cpn* (Tsc/360)));
```

I found that while this worked alright at Face=10, but if I set Face := 100 or 1000, my results were garbage! The simple fix was to change the first part of the expression to read (Face * 1000.0)! I had been sailing past MAXInteger!! (And the System will NOT give you an Error message on this, Only garbaged results!). Note that the same thing can happen with Long Integers. If you set something like this:

```
Long := Integer * 1000
```

The same thing happens. It is best to do this in two steps.

```
Long := Integer;
Long := Long * 1000
```

PREDEFINED IDENTIFIERS.

On page 135 of the Pascal 1.1 Language Manual is a list of Predefined Identifiers. These are *not* Reserved words, but they should be treated as though they were.

You can use these words, such as "str" as variables, but once you have done this, you lose the use of that function! For example, I changed a program to use Long Integers, which, Using the STR function and other String handling built-ins, will enable you to print numbers as $10,987,654.395. But -- NOT if you have a variable "Str", or "Length". You will get a somewhat confusing compiler error message at the point you first invoke the function: "error in variable". The message is actually correct, but when you see it by a function, it is confusing! In the case of such a message, if all else looks OK, check your list of variables!

PRINT FORMATTING.

Although one can format the way reals appear in output by using the suffixes as in ' RealVar:5:2 ', where 5 is the length of the string, and 2 is the number of decimals, This does not work satisfactorily when you are working with large numbers. At that point, it is time to work with Long Integers, which are well documented in the Language Manual. Note also the sections covered under String Intrinsics that show how to convert Long Integers into strings, putting decimal points wherever you may want them, etc. You should look at routines such as KYBDSTUFF (On Call A.P.P.L.E. In Depth) (cited in PBooks), where there are routines to convert a string into a numeric value to go the other way.

Though it does not appear so in the documentation, you can format the printing of Strings. It works just like with numbers. Remember that Integers and negative String output of Long Integers, may have one more space than you see (to provide for a '-', if needed).

MISCELLANY.

Among the limitations of Apple Pascal, one that is most irritating is the 38 block file limit of the Editor. There are several ways to work around this, such as using the Include file option in the Compiler. (This can be a useful step in testing some code before setting it up as a separately compiled Unit.) An easier, but not cheap way is to replace the System.Editor with the Advanced System Editor (from Volition Systems, Del Mar, CA). There are many other benefits to this besides file size. I use it; take a look.

Both the Apple Editor and ASE (see above) have a handy little item that I didn't notice until I had been working with Pascal for over a year. If you look at the first character on the Command line, you will notice that it is ">". Now try changing direction in the Editor by either using + or - or the > < keys. Cute, eh? Handy, too. Note that the >< keys are NOT case sensitive.

Do check out the syntax in the demo programs on APPLE3: particularly DISKIO, which is good on both file I/O and the two binary tree files, Tree and Balanced, which along with DISKIO, give a good view of the use of pointers.

There are good Public Domain programs to transfer files between DOS and Pascal. I use Huffin and Puffin from the Call A.P.P.L.E. in Depth disks. Huffin is in Applesoft and is slow. If you use it, cold-boot your Apple before trying to do anything else, even save a file to disk, as the program does not clean up after itself!

A primary resource for improving one's Pascal techniques has to be USUS (The USCD Users Society) Their dues are $25 per year And a Great Bargain, particularly if you can access their MUSUS bulletin board on CompuServe. That is where I got 90% of the information and code included in this file. I had been somewhat apprehensive about joining this group, since I was afraid that they were all hard-core advanced Pascal programmers, and that my queries might be resented, or the answers would pass right over my head!

I was partly correct. Some of the members have forgotten far, far more about Pascal than most of us will ever know! However, They are most helpful and take pains to make sure that the asker of a question really understands the answer, often taking the time to make sure that they are being clear or leaving code to illustrate the point.

One of the real pluses for an Easterner, is the ability to leave a summary of the problem that had kept me up till the wee hours, just before I crashed, with a good shot at finding answers when I got up the next morning!! (For westerners, I guess, the answer would be there when they got home from work!)

Some readers have been misinterpreting these files to the effect that I am an expert on Pascal. That is not the case.

There are some very experienced people who have been kind enough to support this effort. Their names appear below.

Special thanks to the following contributors: Arley Dealey, David Ramsey, John Stokes, John Baxter, Chris Jewell, Bob Peterson, Randy Bush, Eli Willner, Dennis Cohen, Jim Merritt, Jim Thompson, and all those to numerous to mention, who yelled "Stop !", when I was going astray.

SUPPLEMENT ON BOOKS AND SOURCES.

Since the last publication of the Guerilla Guide, the following sources of information have come to light.

Jim Merritt's excellent column in Softalk has been very useful for me, as he often approaches difficult points a little differently than others. This is worth buying back issues to get, but it will come out as a book in 1984.

While not a text, you might also wish to take a look at the Link Sampler, which is a group of well documented programs aimed at helping you learn Apple Pascal. The disk and documentation are not cheap at about $60, but are well done.

Other Ideas from Dennis Cohen:

"For Standard Pascal, I would recommend *Introduction to Pascal (2d ed.)* by Welsh and Elder.

For Apple Pascal Examples, The two best I have seen were both written by Tom Swan and published by Hayden:
Pascal Programming for Business
Pascal Programming for Games and Graphics

There are excellent examples in these books, but they are both at least at the level of Lewis, so the user should go through the level 1 stuff first."

And, From Arley Dealey:

"You should run, not walk to your book-seller and get *Advanced Pascal Programming Techniques* by Paul A. Sand (Osborne/McGraw-Hill, 1984, ISBN 0-88134-105-3). I've got my differences with the author in some places (some of them quite strong differences) but over-all I think this is definitely the best second text on Pascal I've seen yet." (I have just gotten this, and while I have not yet gotten 'down and dirty' in it, I am quite excited by it. It leads to real-world uses, and has what appear to be the most painless way into file structures and use of pointers I have yet seen! I have a *lot* of respect for Arley's advice! BT)

Intermediate UCSD Pascal Topics

By: **Robert W. Peterson**

SUBJECTS TO BE COVERED

- Pointers and their uses
- File I/O, including declaring and using various file types
- Separate compilation (units)

Variables declared in the VAR part of a procedure or function are allocated memory space in what is called a "stack". Each time a routine is referenced, a new area of the stack is allocated for that procedure's variables. Each routine's variables are allocated at the "top of stack", that is, at the next available location. When a routine terminates, the stack storage allocated is thrown away.

In Pascal, PL/I, Modula-2, and some other languages there is a second area in which variables may be allocated and this second area is called the "heap". Variables are allocated on the heap using explicit requests instead of the implicit request of calling a routine. Stack variables exist only until the routine containing their declaration terminates. Heap variables exist until explicitly removed.

The routine doing the heap allocation returns the location of the allocated memory so that the area can be referenced. In Pascal the variable returned is placed into something called a pointer variable or, more commonly, a **pointer**.

A pointer, like any Pascal variable, is able to represent only a specific type. That is, a pointer to an integer cannot point to an array.

MEMORY ALLOCATION

VAR variables are allocated in the stack when a routine is entered and deallocated when the routine exits.

Heap variables are allocated explicitly by the programmer and remain allocated until explicitly deallocated.

Memory references

VAR variables are referenced as a displacement from the beginning of a stack frame.

Heap variables are referenced using a **pointer.**

Pointers behave very much like machine registers: They provide a baseline from which fields are referenced with base/displacement addressing.

The only difference between a pointer declaration and any other type or variable declaration is the addition of an up arrow or caret. For example:

```
VAR
     a : integer ;
     b : ^integer ;
```

The declaration of **a** will cause an integer to be allocated space on the stack. The declaration of **b** will cause a pointer to be allocated space on the stack but will cause *no* allocation of space for an integer.

A more interesting example of a pointer declaration is:

```
TYPE
     ptr_index = ^index_type ;
     index_type = RECORD
          key : STRING[11] ;
          recnumber : integer ;
          next : ptr_index ;
          prior : ptr_index ;
          end ; (* index_type *)
VAR
     ptr : ptr_index ;
     index_anchor : ptr_index ;
```

Several new ideas are illustrated above:

The declaration of ptr_index uses index_type *before* index_type is declared. This kind of forward reference is allowed only in pointer declarations.

Next and prior are pointers to index_type; that is, index_type contains pointers which point to index_type.

In UCSD Pascal implementations prior to Version IV.0, the implementation of the heap looks much like a manually managed stack. When using this "stacklike heap", the programmer inserts a place marker into the heap using a standard procedure called MARK. One or more allocations are made on the heap using NEW. (The amount of

space required by each allocation is determined by the compiler.) To recover the space taken up by the NEW allocations since the last MARK, a **RELEASE** is issued. Using the stack-like heap, deallocation of a single heap entity is not possible.

In UCSD Pascal Version IV.0 and above, a standard heap implementation is available, as well as the stack-like heap with its MARK and RELEASE. To deallocate a single heap element in Version IV.x, **DISPOSE** is implemented.

MARK and RELEASE take as an argument a pointer to an integer. The argument passed to NEW and DISPOSE is a pointer variable.

Like other Pascal variables, pointers are not automatically initialized. Unlike other Pascal variables, uninitialized pointers can have disasterous results, such as corrupting the system and causing a system crash.

Thus the skeleton of an example of the use of pointers is:

```
TYPE
     ptr_index  = ^index_type ;
     index_type = record
          key        : string[11] ;
          recnumber  : integer ;
          next       : ptr_index ;
          prior      : ptr_index ;
          end ; (* index_type *)
VAR
     ptr          : ptr_index ;
     index_anchor : ptr_index ;
     marker       : ^INTEGER ;
BEGIN
     ptr          := NIL ;
     index_anchor := NIL ;
     MARK( marker ) ;
     use_the_pointer( index_anchor ) ;
     (* to be defined *)
     RELEASE( marker ) ;
END ;

PROCEDURE use_the_marker( var anchor : ptr_index ) ;
BEGIN
     NEW( anchor ) ;
     anchor^.next      := NIL ;
     anchor^.prior     := NIL ;
     anchor^.key       := ' ' ;
     anchor^.recnumber := 0 ;
END ; (* use_the_marker *)
```

This procedure simply allocates on the heap something of type "ptr_index", returns the location of that allocation in the pointer variable "anchor", and initializes the record now on the heap. The first part of the initialization sets the two pointers within the record to NIL, that is, to having no valid value. The latter part sets the string to an empty string and the record number to zero.

Pointers can be used to solve several common programming problems:

The UCSD Pascal compiler uses the heap to build the symbol table for each procedure it compiles. As each routine's variables are allocated, they are placed on the heap after the older variables already stored there. An identifier is looked up by searching the heap, beginning with newest entries and moving toward the entries for the oldest procedure. This way the scope of identifiers is simple to maintain.

The heap can be used to contain a list of items, much like an array, without requiring prior knowledge of the number of items in the list. A cross reference program is a good example. The number of variables as well as the number of references to each are not known ahead of time. Using the heap allows a limited amount of self-configuring: programs with a large number of variables but limited references and programs with a small number of variables with lots of references both have use of the entire heap, but use the space in different ways.

Trees, allocated on the heap, are frequently used by indexing schemes for storage of keys. One method of sorting uses the heap for intermediate storage.

UCSD Pascal supports several different types of files:

TEXT
INTERACTIVE
TYPED
UNTYPED

In addition, UCSD Pascal supports random access of typed and untyped files.

We will not discuss UNTYPED files.

IORESULT is an integer-valued function. The value of the function reflects the termination status of the most recent I/O operation.

IORESULT's purpose is to allow the programmer to monitor the state of an I/O operation. However, this is possible only when combined with the compiler's option that turns off the automatic result checking normally inserted

automatically:

```
(*$I-*)
file operation
(*$I+*)
if IORESULT <> 0 then error_procedure ;
```

Later on there will be some examples of how IORESULT might be used.

In order to do manipulate information using a file, the file must first be made ready for use, or opened. Pascal does this with the RESET and REWRITE verbs.

RESET opens a file which already exists. Parameters to RESET are the name of the file as declared in the VAR part of the block, and, optionally, a string containing the name of the disk file to be accessed. RESET without the string parameter will "rewind" an already open file, i.e., the next record read will be the first record of the file. A RESET will fail if the specification in the string is not valid.

REWRITE creates a new file on disk and initializes it for output. Like RESET, REWRITE takes as its first parameter the file name declared in the VAR part of the block. Its second parameter is a string containing the name of the disk file to be created. If the file already exists, the REWRITE will delete the existing file and create a new (empty) file with the same name.

CLOSE indicates to the system that access to the file is no longer required. CLOSE takes one or two parameters. The name of the file as declared in the VAR area is required. The second parameter is the type of close needed and is optional.

If the close type is not specified, a **NORMAL** close is done. That is, CLOSE simply sets the file state to closed. If the file was opened using REWRITE and is a disk file, no entry is made in the disk directory and the file is not saved. A file opened with RESET remains after a NORMAL close.

The **LOCK** option will cause the disk file associated with the open file to be made permanent in the directory if the file is on a block- structured device and the file was opened with a REWRITE. If the file was opened with a RESET, a NORMAL close is done.

The **PURGE** option removes the file from the directory, effectively deleting the file.

The **CRUNCH** option LOCKs the file and truncates it to the point of last access. "The point of last access" is the position in the file of the last I/O operation.

CLOSE on a closed file causes no action.

A **TEXT** file is a file of character divided into lines by end of line characters. This type of file is suitable for input to the various system programs such as the text editor and the Pascal compiler.

In addition, a UCSD Pascal TEXT file has an internal structure:

The first two blocks are used by the Editor.

Characters are stored in block pairs: Each block pair ends at a line boundary.

Any extra bytes are filled with nulls (binary zeroes).

When the file is opened, an initial READ is executed.

An **INTERACTIVE** file is a special case of a text file. The standard files **INPUT, OUTPUT,** and **KEYBOARD** are of type INTERACTIVE and are automatically associated with the display/keyboard devices. In general, applications will not need to declare any INTERACTIVE files and if they do, INTERACTIVE files behave much like TEXT files. The major exception is that there is no implied READ when an INTERACTIVE file is opened. There are also minor differences in how the EOF function behaves with an INTERACTIVE file.

For example:

```
VAR
    a_text_file : TEXT ;
    crt : INTERACTIVE ;
    listing : TEXT ;
BEGIN
    RESET( CRT, 'CONSOLE:'  ) ;
    REWRITE( a_text_file, 'peterson.text'  ) ;
    (* #6: is a reference to the printer device *)
    REWRITE( listing, '#6:'  ) ;
END ;
```

In this example, file **CRT** is associated with the system console, **a_text_file** is linked to a file named 'peterson.text', which is stored on the default disk drive, and **listing** is a file associated with the printer port.

TEXT and INTERACTIVE files are accessed for input using the READ and READLN verbs.

During the following discussion assume the following declarations:

```
VAR
     ch          : CHAR ;
     i, j, k, l  : INTEGER ;
     r, q        : REAL ;
     s           : STRING ;
```

READ(ch) will read from the keyboard the next character and display it on the console. This is how the system reads commands without requiring a carriage return.

READ(s) will read characters until return is typed but will not include the carriage return character.

READ(i) will expect an integer constant followed by a space. Anything other than a digit or space as the initial character will cause a run-time error. After the first digit is found, any non-digit will cause the input of the integer to terminate.

READ(i, j, k) will behave as above but will read three integers instead of one.

READ(r) will read a real constant, with restraints similar to reading an integer.

READLN, which stands for "read a line", causes the program to ignore all input until a carriage return is typed.

```
   READLN( i, j ) ;
is equivalent to
     READ( i ) ;
     READ( j ) ;
     READLN ;
```

Both READ and READLN take as an optional first parameter a file id. When the file id is omitted, the default file INPUT is assumed.

```
     READLN( i, j ) ;
is assumed to be
     READLN( INPUT, i, j ) ;
```

since INPUT is the default input file.

Two Boolean functions are available for use with text files: **EOF** and **EOLN**. Both these functions take as an optional parameter a file id. If the parameter is left off, the standard file OUTPUT is assumed.

EOF stands for "End Of File". EOF is true when the end of file character is read from the keyboard or the end of a disk file is reached. Attempting to read beyond the end of a file is a run-time error.

EOLN is true whenever a READ(INPUT) is terminated by a carriage return or, for a disk file, when the end of line character is encountered. Doing a READ(CH) when EOLN is true places a space into CH, not an end of line character or carriage return.

The **PAGE** verb can be applied to TEXT and INTERACTIVE files. The parameter is a file id. The PAGE verb causes a FormFeed character to be output. On many printers this will cause the printer to slew to the top of the next page. On some terminals, PAGE will cause the screen to clear.

Where TEXT files allow only characters to be read and written, typed files allow any type of data to be read and written and, in addition, allow random access to any record in the file. Typed files are, however, useful only for files stored on disk.

The RESET, REWRITE, CLOSE, and EOF routines function with typed files just as they do with TEXT files.

For example:

```
PROGRAM typedio ;
TYPE
     t_str = string[40] ;
     t_nad = record
          name               : t_str ;
          addr1, addr2, city : t_str ;
          state              : STRING[2] ;
          zip                : STRING[9] ;
VAR
     name     : t_str ;
     nad_file : file of t_nad ;
     nad_rec  : t_nad ;
BEGIN
     RESET( nad_file, ' name.addr.data'  ) ;
      . . .
     CLOSE( nad_file, lock ) ;
END. (* PROGRAM typed *)
```

Input and output of a typed file's records uses the concept of a "window" variable. The window variable allows the program to see a single "scene" or record of the entire file. The window variable in Pascal is the declared name of the file, in our example, "nad_file".

To reference the record currently in the window variable, the file variable is used as if it was a pointer variable. For the previous example,

```
     name          := nad_file^.name ;
     nad_file^.zip := ' 74075'  ;
```

To make a record available at the window, Pascal supplies the GET verb. Get takes as its single parameter a file id.

To place the current contents of the window variable into the file, Pascal supplies the PUT verb. PUT also has as its single parameter a file id.

Using the above declarations, the following is a valid sequence of statements:

```
GET( nad_file ) ;
READLN( name ) ;
nad_file^.name := name ;
READLN( nad_file^.addr1 ) ;
PUT( nad_file ) ;
```

The above fragment would load the window variable with the contents of the next record of the file, change two fields of the record and place the record back into the file in the same place. (This assumes the file already existed and was RESET.)

Each GET reads the next record of the file. Each PUT writes the next record of the file. The GET and PUT locations for the same file are independent of each other. Doing a GET when EOF is true results in an error, but a PUT is valid *if there is room on disk to write the record.*

UCSD Pascal provides **SEEK** to randomly position the GET and PUT locations within a file. SEEK takes as parameters the file id and an integer which specifies the record number to be located. The next GET will fetch that record and the next PUT will replace that record. A GET or PUT *must* executed between every pair of calls to SEEK. SEEK sets to false EOF and EOLN for the referenced file.

Records in a file are numbered, for the purposes of SEEK, from zero.

Note that SEEK works only for typed files. SEEK is not legal for files of type TEXT or INTERACTIVE.

A good example of using the IORESULT function is how to open a file without crashing the program if the open fails. The following code fragment illustrates this:

```
REPEAT
     READLN( file_name ) ;
     (*$I-*)
     RESET( phyle, file_name ) ;
     (*$I+*)
     result := IORESULT ;
     if result <> 0
          THEN
               BEGIN
               REWRITE( phyle, file_name ) ;
               result := IORESULT ;
               END
          ELSE
               BEGIN
               result := 1 ;
               WRITELN( file_name, ' exists!' ) ;
               END ;
UNTIL result = 0 ;
```

The loop will exit only when a new file is opened **and** the new file will not overwrite an existing file.

Many times a programmer or group of programmers will wish to develop an application in sections. This may be required if the application is too large to be compiled as a single chunk because of space or time constraints or if different programmers are assigned different portions of the project.

Or they may wish to develop a number of tools for use across several applications. Such a tool might be a formatted screen utility, file access routines, a math package, or a database manager.

In the UCSD Pascal language environment separate compilation is accomplished by using a language construct called a **UNIT**. A UNIT is made up of two parts: an **INTERFACE** section and an **IMPLEMENTATION** section. A Unit can be compiled by itself, as part of a group of units, or as part of a Pascal program. If it is not compiled as part of a program, the resulting code file is frequently combined with other units into a library.

A separately compiled UNIT might look like this:

```
UNIT thisone ;
INTERFACE
VAR
     device_status : INTEGER ;
FUNCTION keystatus : BOOLEAN ;
PROCEDURE clear_line( line : integer ) ;
IMPLEMENTATION
VAR
     device_address : INTEGER ;
FUNCTION keystatus ;
BEGIN
     (* the body of the function *)
END ;

PROCEDURE clear_line ;
BEGIN
     (* the body of the procedure *)
END ;

END. (* thisone *)
```

Declarations which will be referenced by the program using the UNIT are declared in the

INTERFACE part of the UNIT. Such declarations can include references to other units, constants, types, variables, and routines. Items which occur in the IMPLEMENTATION part are private to the UNIT; they cannot be referenced or used in any way outside the UNIT's IMPLEMENTATION part.

Routines declared in the INTERFACE part are assumed to be declared FORWARD but do not require the keyword FORWARD. This means that the body of the routine cannot be in the INTERFACE part but must be in the IMPLEMENTATION part. It also means that any parameters must be included in the INTERFACE declaration and not mentioned in the IMPLEMENTATION routine heading. In versions prior to IV.0, a unit could not contain SEGMENT (or overlay) routines; this restriction has been eased in Version IV.0 and up. In no version of UCSD Pascal can an INTERFACE routine be an EXTERNAL machine code routine.

Constants, types, variables, and additional routines may be included in the IMPLEMENTATION part if needed. In version IV.0 and up, a UNIT can reference another UNIT in the IMPLEMENTATION part. Prior to Version IV.0, another unit could be referenced only in the INTERFACE section of a unit.

A UNIT is referenced by a program or another unit with the USES keyword followed by a list of one or more unit names:

```
PROGRAM testunit ;
USES thisone ;
VAR
    prog_variable : INTEGER ;
BEGIN (* testunit *)
 .
 .
 .
END.
```

The effect of the USES keyword will be as if the program text looked like this:

```
PROGRAM testunit ;
VAR
    device_status : INTEGER ;
FUNCTION keystatus : BOOLEAN ;
PROCEDURE clear_line( line : integer ) ;
VAR
    prog_variable : INTEGER ;
BEGIN (* testunit *)
END.
```

Observe that if the unit had had a variable "prog_variable" the compiler would have detected a duplicate declaration error when it encountered the second "prog_variable". Also note that the ordering restrictions for declarations are relaxed when a unit causes declarations to be copied.

When a UNIT is referenced from another unit's INTERFACE section, the compiler requires that the program reference *both* units and do so in the correct sequence. If the unit "secondone" references the unit "firstone", then the program's USES phrase looks like:

```
PROGRAM unittest ;
USES firstone, secondone ;
VAR
    prog_variable : INTEGER ;
BEGIN
END.
```

The default library in which the compiler attempts to find the named unit is **SYSTEM.LIBRARY.** If the code of "firstone" is in the library FIRSTLIB.CODE, the code of "secondone" is in WORKLIB.CODE, and "os_stuff" is in SYSTEM.LIBRARY, then the option library specification of the USES clause is used:

```
USES
    (*$U FIRSTLIB.CODE *) firstone,
    (*$U WORKLIB.CODE *) secondone,
    (*$U SYSTEM.LIBRARY *) os_stuff ;
```

At compile time, the compiler assumes all units are stored in *:SYSTEM.LIBRARY, and searches there for units' INTERFACE sections. The compiler does not automatically search any other library if a unit cannot be found.

The compiler **U** option is available and allow the programmer to specify an alternate library of units.

The compiler always searches in the most recently named library, or if none has been specified, in *:SYSTEM.LIBRARY.

Multiple libraries can be named in a USES clause, and/or multiple units can reside in a single library.

```
USES
                                a,
(*$U #9:firstlib.code *)  c,
(*$U    worklib.code  *)  g, b,
(*$U *:SYSTEM.LIBRARY *)  os_stuff ;
```

Version IV.0 and later offer a variation on the USES clause called **selective USES.** Sometimes a compile will get so large in terms of the number of symbols that the compiler cannot successfully complete the compile. To relieve this problem

when a large number of units are referenced, the compiler allows only parts of a unit to be included in the symbol table.

For example:

```
USES ScreenOps( SC_CLR_SCREEN )
```

will cause only the identifier SC_CLR_SCREEN to be entered into the symbol table, even though the unit's INTERFACE section contains many other identifiers. This capability is especially useful for programs that make use of the IV.0 KERNEL unit.

A program must reference units in a specific sequence and must reference all units referenced in unit INTERFACE sections.

A unit that does not reference another unit in its INTERFACE section does not need to be in any particular sequence in the program's USES clause.

A unit that references another unit in its INTERFACE section requires the program to order the units in the program's USES clause. Each unit must be listed in the program's INTERFACE clause before another unit references it. This is actually the old "define before use" rule. For example:

Assume unit "secondone" references unit "firstone" in secondone's INTERFACE section. Then the program's USES statement must be

```
USES firstone, secondone;
```

In order to execute a program which uses one or more separately compiled units, UCSD Pascal versions prior to version IV.0 required the unit's code must be linked with the main program's code. IV.0 and up do not allow or require units to be linked prior to running the program. Version II Units are linked using **Linker**, the system's link editor.

If all the units referenced are stored in *:SYSTEM.LIBRARY, the program may be R(un. R(un will automatically invoke the Linker before executing the program.

If one or more of the units referenced are not in *:SYSTEM.LIBRARY the Linker must be explicitly invoked and the required libraries named.

The Linker is invoked by typing an L at the system's main command prompt. The Linker will ask first for the file containing the main program's code. Following that the Linker will prompt for the library files to be used to resolve references to units.

Using the above units and assuming that the main program has just been compiled (and as a result is in *:SYSTEM.WRK.CODE), the interaction with the Linker will look like this (typed in information is underlined):

```
Host file? testunit
Opening *SYSTEM.WRK.CODE
Lib file? firstone
Opening firstone.CODE
Lib file? worklib
Opening worklib.CODE
Lib file? *
Opening *SYSTEM.LIBRARY
Lib file? (return)
Map name? printer:
Reading ...  Here the Linker writes as it works
Output file? (return for *:SYSTEM.WRK.CODE
        or a file name ending in .CODE)
.
.
.
```

A response to "Host file?" of carriage return will cause Linker to use the work file as its input file. The response to "Host file?" must be a code file containing a program. A code file containing only units is not a valid response.

Responding to "Lib file?" with an asterisk is a shorthand way of specifying *:SYSTEM.LIBRARY.

Responding to "Output file?" with only a carriage return causes the Linker to place the linked output file into *:SYSTEM.WRK.CODE. A file name is a legal response but it *must* end with ".code". The Linker does *NOT* append ".code" and a file created without the ".code" suffix is not a code file!

Jensen, K. and Wirth, N., *Pascal User Manual and Report*, Springer-Verlag, 1975, 167 pages.

Wirth, N., *Algorithms + Data Structures = Programs*, Prentice-Hall, 1976, 366 pages.

Ledgard, H.F., Hueras, J.F., and Nagin, P.A., *Pascal with Style: Programming Proverbs*, Hayden, 1979, 210 pages.

Clark, R., and Koehler, S., *The UCSD Pascal Handbook*, Prentice-Hall, 1982, 356 pages.

Fast 2-D List Processing

By: **Jai Gopal Singh Khalsa**

LIFE Revisited - A Fast Two-dimensional

List Processing Technique

"Conway's Game of Life is a simple rule for successive populations of a bitmap. The rule involves the neighbor count for each cell - how many of the eight adjacent cells are occupied. Each cell will be occupied in the next generation if it has exactly three neighbors, or if it *was* occupied and has exactly two neighbors." (*Smalltalk-80 The Language and Its Implementation*, p. 412)

I did my first implementation of the LIFE game when reading the Smalltalk-80 book during the summer of 1983. I was fascinated by the idea of a bitmap as a Pascal record (packed array[0..79,0..23] of boolean); it seemed vaguely relevant to the USUS discussions going on at that time about windowing algorithms... and it was the first example in the Smalltalk book that I could actually play with using UCSD Pascal and my no-graphics CRT. Although the particular Smalltalk-80 example algorithm was not at all applicable, the ideas about object-oriented programming presented there contributed strongly to the current implementation.

The game starts with a blank screen and the prompt line:

(move cursor) S)et cells, B)egin, <ESC> to quit ?

Typing 'S' causes the cell under the cursor to be toggled between '.' (empty) and '@' (occupied). When a satisfactory pattern has been created, the 'B)egin' option invokes a continuous loop that computes the resulting pattern for the next 'generation' based on the current pattern and Conway's rule.

That first program actually visited every cell on the screen and counted its neighbors; straightforward but very laborious. Next I narrowed it down somewhat by flagging only those columns and rows where an occupied cell was known to exist along with its two adjacent columns and two adjacent rows to be checked for possible 'births' in the next generation. Still rather slow but it worked and I tired of the game and left it at that.

Improving the Algorithm:

During the summer of 1984, Mel Saffren on MUSUS asked for LIFE implementations and so I dusted mine off and uploaded it to him. He complained it was slow and uploaded his own version which was five times faster than mine! Also, his had wrap around, reflect, and two-sided plane topologies where I had just let things die as they went off the edge of the screen. His implementation of the various topologies, however, was treated as an exceptional condition and caused a very noticeable slow-down. This was particularly evident when a single crawler was racing across the screen and then hit the edge, slowing to a tedious pace. (NOTE: see below for explanation of the 'crawler' pattern.)

The significant features of Mel's implementation (besides the various topologies) were:

a) he visited the occupied cells and incremented the neighbor count of all eight adjacent cells. Then he looked for cells whose neighbor counts fitted Conway's rule for Life. This in itself was a lot less work than actually counting neighbors for *every* cell (at 8 operations per).

b) he limited his search by keeping a 'boundary' record which held the leftmost, rightmost, topmost, and bottommost positions of occupied cells (updated as births and deaths were computed). This defined a constantly varying rectangle which had the effect of speeding the search when the area was small but dramatically slowing it when the population spread across the screen. This was true even when there were few occupied cells as is the case when you have two crawlers on opposite sides of the screen.

The first of these features was the crucial one as I already had a method for narrowing the search for occupied cells. As an example of its effect, consider a screen pattern who's rectangular boundary is 15 by 40 (600 cells) where only 30 of the 600 are actually occupied. Counting the neighbors of all 600 cells requires 600 times 8 'if' statements or 4800 operations plus the additions performed when the 'if' statements are TRUE (another 240 operations?) for a total of 5040 operations. Incrementing the neighbor count of all eight cells adjacent to occupied cells requires 600 'if' statements (to see if occupied) plus 8 additions for each of the 30 occupied cells for a total of only 840 operations. This ratio alone (5040 to 840) may

account for the five-fold increase in speed of Mel's implementation over my original. In both cases, the entire rectangular area must be re-examined after the neighbor counts are computed to apply Conway's rule.

The Fast List Algorithm:

It became evident that the next major improvement in the algorithm would be to further limit the time spent counting neighbors and testing for births/deaths in those cells that couldn't possibly be affected by the previous generation. A super-fast two-dimensional list was implemented using a 'listObject' to keep track of cells within a column and a 'graphObject' to keep track of 'listObjects' (i.e., columns) that contain cells.

The main advantage of the 'listObject' and 'graphObject' approach is that I visit only occupied cells rather than all those in a rectangular space; in the process of incrementing the neighbor count of all adjacent cells, I construct a second graphObject, 'toExamine', that consists only of those cells that have three (or more) neighbors plus all those that were occupied before. I then quickly 'traverse' this two-dimensional list and apply the rule of Life. If the new value is different than the previous generation, i.e., the current state of the bitmap, I update the screen (liveChar or deadChar) and bitmap and add or subtract the cell to/from the 'occupied' graphObject list. The effect is that I can have two crawlers 'racing' around the screen at a speed independent of their screen position and proportional only to the number of occupied cells on the screen. Also, wrapAround and reflect are treated as a normal function of range checking and so there is no slow-down when the edges are encountered.

The two-dimensional list was implemented using the following record structures:

```
CONST xLow = 0;
      xHigh = 79;
      yLow = 0;
      yHigh = 22;

TYPE listObject = record
       numCells : 0..24;
       cellList : packed array[1..24] of yLow..yHigh;
       cellIndex: packed array[yLow..yHigh] of 1..24;
     end;

     graphObject = record
       numCols  : 0..80;
       colList  : packed array[1..80] of xLow..xHigh;
       colIndex : packed array[xLow..xHigh] of 1..80;
       colObject: array[xLow..xHigh] of listObject;
     end;
```

The 'numCells' field of listObject indicates how many cells are in 'cellList'. Zero indicates that the object (i.e., column) has no cells in its list; otherwise, the elements 1..numCells of cellList comprise the list of cells (i.e., rows) contained in the object. The 'cellIndex' field is used for speedy location and removal of a cell (i.e., row) from the cellList array.

'graphObject' is identical in function to listObject except that it maintains a list of listObjects (i.e., columns) rather than cells (i.e., rows). Their use can best be understood by looking at the following two procedures that are used to maintain the two-dimensional list. Note that 'cellIndex' and 'colIndex' are maintained by the 'cellAdd' procedure but are actually *used* only in the 'cellRemove' procedure. These indexes allow the element being removed to be directly replaced by the last element of the list without any time-consuming search for the element being removed. This is an excellent example of the speed vs. memory tradeoff that one often encounters in programming; the penalty of sixteen extra bytes in the listObject structure (nearly doubling its size) is compensated by the fact that you always know exactly where in cellList to find a particular cell. If memory is more critical than speed, a linear search of cellList will tell you the same thing.

```
PROCEDURE cellAdd(col,row: integer;
                  VAR OneGraph: graphObject);
begin
  with OneGraph.colObject[col] do
    begin
      numCells := succ(numCells);
      cellList[numCells] := row;
      cellIndex[row] := numCells;
      if numCells = 1 then with OneGraph do
        begin
          numCols := succ(numCols);
          colList[numCols] := col;
          colIndex[col] := numCols;
        end;
    end;
end; {cellAdd}


PROCEDURE cellRemove(col,row: integer;
                     VAR OneGraph: graphObject);
begin
  with OneGraph.colObject[col] do
    begin
      cellIndex[cellList[numCells]] := cellIndex[row];
      cellList[cellIndex[row]] := cellList[numCells];
      numCells := pred(numCells);
      if numCells = 0 then with OneGraph do
        begin
```

```
        colIndex[colList[numCols]] := colIndex[col];
        colList[colIndex[col]] := colList[numCols];
        numCols := pred(numCols);
      end;
  end;
end; {cellRemove}
```

Traversal of the two-dimensional list is accomplished as follows ('toExamine' is of type 'graphObject'):

```
with toExamine do for cPtr := 1 to numCols do
  with colObject[colList[cPtr]] do
    for yPtr := 1 to numCells do
      CheckCell(cellList[yPtr]);
```

Note that neither cellList nor colList are ordered lists; consequently, screen updates appear haphazard.

Performance Enhancement:

Closely intertwined with a good algorithm are the implementation details that affect speed. The 'cellIndex' and 'colIndex' are examples of this. While they improve the ability to quickly maintain the 'occupied' graphObject (where cells are added *and* removed), they actually slow down the process of creating the 'toExamine' graphObject since, in the current version of the program, cells are never removed from the toExamine list; the overhead of maintaining these indexes is therefore a waste of time.

In addition to the details of the list algorithm, other details have important effects. At first the various topologies (offEdge, wrapAround, reflect) were done as part of the bounds check within the incrNeighbor procedure. It was then a simple matter to call that proc eight times for each occupied cell:

```
incrNeighbor(pred(x),pred(y));
incrNeighbor(     x ,pred(y));
incrNeighbor(succ(x),pred(y));
incrNeighbor(pred(x),     y );
incrNeighbor(succ(x),     y );
incrNeighbor(pred(x),succ(y));
incrNeighbor(     x ,succ(y));
incrNeighbor(succ(x),succ(y));
```

This, though, requires a minimum of 4 'if' statements within incrNeighbor. I realized that by moving the topology algorithm out of incrNeighbor, x need only be checked once per *column* and likewise, pred(x) and succ(x) need only be computed once per column. Similarly, pred(y) and succ(y) need only be computed once per cell rather than three times and if they were off the edge of the screen, one 'if' statement would suffice where three were required before. The resulting code looks a bit messier but is significantly faster. It was discovered later, however, that another interesting topology, the Klein bottle (wrapAround with a twist) does not apparently lend itself to these same shortcuts although is quite easy if done within incrNeighbor. The solution is to have two (or more) different procedures to do the calls to incrNeighbor: 'PROC fastWrap' would be as is found now in the code and 'PROC KleinBottle' would be as above with its own version of incrNeighbor to apply the topology algorithm *within* the incrNeighbor procedure.

The LIFE Benchmarks:

In order to evaluate the performance of various algorithms and program enhancements, benchmarks are required. I used the two patterns below:

```
Crawler  Smalltalk
  ■■■      ■■
   ■       ■■
  ■         ■
```

The term 'crawler' is my own and simply describes the apparent behavior of that pattern. It takes four generations to cycle back to its original shape and in the process, moves diagonally one cell. Note that 0 & 2 and 1 & 3 below are identical; they are simply rotated 90 degrees on the surface of the plane and 180 degrees in 3D. The aspect ratio of the screen disguises this fact.

```
                ■       ■■      ■■      ■■■
          ■■■   ■■    ■ ■       ■■       ■
            ■   ■ ■     ■       ■        ■
           ■

generation 0      1       2       3       4
```

I call the other one the Smalltalk pattern because it was suggested in the Smalltalk-80 book. An interesting symmetrical pattern I have discovered that goes a long time with wrapAround is the following:

```
■■ ■■
■   ■
 ■■■
```

The benchmarks now stand as follows on my Sage (thanks to Mike Berg for the NCI p-System and TURBO Pascal times on his IBM PC):

Test (wrapAround)	Time	Generations JGS	Mel	Ratio JGS/Mel	
Single crawler	1 minute	771	622	1.24	
Two crawlers (A)	1 minute	477	62	7.69	(!)
" " (B)	1 minute	477	108	4.42	(!)
" " (A)	3 minutes	1425	277	5.14	(!)
" " (B)	3 minutes	1425	203	7.02	(!)
Smalltalk pattern	3 minutes	203	173	1.17	

Smalltalk pattern, no wrapAround, first 50 generations;
JGS = 14.4 seconds Mel = 12.6 seconds JGS/Mel = 1.14 (drat!)

NOTE: The two crawlers were separated vertically by half a screen to avoid collision, one going upLeft and the other upRight. Test (A) started the crawlers on opposite sides of the screen while test (B) started them in the middle, one above the other.

Commenting out the screen updates (per Randy Bush's suggestion), I got the benchmark times below (column headed 'no I/O'). The program is certainly pushing on the I/O barrier and, contrary to my expectations, more so when the screen is heavily populated. As the LIFE algorithms improve(?), testing without I/O will be a must.

Test (wrapAround)	Time	Generations with I/O	no I/O	%I/O
Single crawler	1 minute	771	926	17%
Two crawlers	3 minutes	1425	1796	21%
Smalltalk pattern	3 minutes	203	256	21%
Smalltalk pattern, no wrapAround, first 50 generations (seconds)		14.4	10.6	26%

Conclusion:

Some may be interested in the LIFE game for its own sake, for the effects of the various topologies, and as a stepping stone into the larger field known as 'cellular automata'. My own fascination is with the opportunity it gave for exploring, in a very limited way, the object style of programming inspired by Smalltalk. In that language, an object is a combination of data and the methods appropriate for operating on that data. Within UCSD Pascal, a closer representation would be a unit designed to operate on the listObject type and allow 'instances' of listObject to be created. Additional features (besides 'add' and 'remove') such as 'cellGet', 'cellPut', 'listSort', 'cellNext', etc., could be added and would be especially useful if they could handle variable size data objects (records). For example, it is often the case in applications programming that a list of one hundred items (0..99) is needed for payment and shipping methods, transaction types, etc. A unit designed to handle such a list with the record size specified at runtime would be very handy.

This, of course, is way beyond the needs of a speedy LIFE implementation but I hope the germ of the idea is sufficiently illustrated here to stimulate further exploration of the object-oriented style of programming abstraction. The source code for this version of LIFE can be found in the MUSUS XA3 database as LIFE.TXT and will hopefully find its way into the USUS software library. In any case, enjoy LIFE!

Apple /// Pascal Review

By: **Dennis Cohen**

Apple /// Pascal 1.2 is a rather interesting milestone in the history of the Version II p-System. Those of you familiar with the Pascal for the Apple /// are aware that there is a Version II system which gives most of the advantages ascribed to the Version IV system without sacrificing the advantages of Version II. With Apple /// Pascal 1.2, Apple has come even closer to an idyllic blend.

There are two Pascal compilers available under 1.2, the native 1.2 compiler and one labelled as 2.0. The 2.0 compiler will be discussed later as it merits quite a bit of verbiage.

The first thing that I noticed about 1.2 was the user interface. The last program executed (or edited) is remembered and just about everything has a default up with the prompt which you can edit. One of the nicest features of ASE (which still doesn't exist for the ///) is the way you can do a directory list when you forget a filename. Well, with the 1.2 Editor, you get a similar capability via a pop-up window on the right side of the screen (similar to that available via System Utilities). This same interface is available at the X(ecute prompt, the C(ompile prompt, the A(ssembler prompt, and many others. About the only thing that doesn't have the new interface is the Filer. I guess this makes sense, since you already have all the wildcard capabilities (and a lot of people rename the System Utilities as SYSTEM.FILER and have the interface there as well). The routines for this

interface are even included in SYSTEM.LIBRARY (as DIRSTUFF), so that you can use it in your programs as well, a very nice touch on Apple's part.

The 2.0 compiler is unique. It not only has the user interface that I mentioned above, it has even more. For the first time the Pascal compiler need not be named SYSTEM.COMPILER, it can be named PAS.CODE (for example) and executed. There is an options file that comes with it that allows you to set all sorts of options at the time you compile without having to embed compiler directives in the source. The ongoing display of progress is a lot more informative than we have been used to up to this time. It tells you what source line of the program it is currently compiling as well as the line of whichever include file is active (and they can now be nested to a depth of three, *a la* Version IV). If there is an error and you select to go to the editor, the system remembers which file was active and goes to the proper line in the proper file. Another nice touch is that by pressing <Escape> during a compile, you are given the option of cleanly terminating the compilation. If you want the compilation to continue to the end regardless of errors, one of the options allows you to have the errors logged in a file and for the compilation to complete. There are two major advantages to this scheme: the first is if you just want a complete list of all the errors that were found and don't feel like typing <space> to continue each time; the second is when there might be just one or two isolated errors and you want to test the unaffected code. Also available are displays of the symbol table and listings that detail the generated p-codes.

The language has also been extended somewhat to get slightly closer to a blend of ISO Pascal and Modula-2. You may now have constant expressions, and may (via a special notation) embed control codes in string constants. Numeric constants may be designated in either binary, decimal, or hexadecimal. While AND and OR work as always, you can now use & and | to denote short-circuit evaluation. The similarity to Modula that I mentioned is in the relaxed order of declaration. You can now group and intermix declarations and procedures to reflect their logical relationships. Those of you who (like me) despise enumerating case label lists can now use subranges (and we still have OTHERWISE). Looping constructs now have two new reserved words related to them, CYCLE and LEAVE, which do what their names indicate. Another similarity to Modula and to ISO Pascal is that you can now, finally, have formal procedure parameters.

The only drawback that I have found to 1.2 is that everything is bigger, which makes for less disk space. The memory space limitations have never come close to biting me on my 256K system; however, my 1.5 MB of disk gets chewed up rather quickly (especially with all of the Toolkit utilities online). Oh yeah, there is one more limitation -- if you use the 2.0 compiler, you cannot generate code to run on the Apple //s. I guess that's what the 1.2 compiler is for.

If all of this discussion hasn't whetted your appetite, I don't know what will.

Hard vs. Soft Interrupts

By: **Jon Bondy**

The following article was written as a result of a series of discussions between the author, Randy Bush, and [indirectly] Bill and Verlene Bonham. The author does not claim these ideas to be his own original thoughts...

The introduction of the concurrency facilities of the III.0 and IV.0 systems provided the p-System programmer with the ability to run multiple processes simultaneously while controlling their synchrony with semaphore variables. The introduction of the Attach facility has allowed synchronization of service processes with external events; with interrupts. This paper discusses some problems which have arisen in the utilization of interrupts under the p-System.

As on all other computers, in the p-System interrupts are serviced at the end of 'instructions'. Because p-System interrupts are really p-codes, p-System interrupts are really pseudo-interrupts; they occur at the end of p-codes, not at the end of physical machine instructions. The interpreter acknowledges an interrupt at the completion of execution of a p-code, but not during that execution. Some p-codes, such as UnitRead and UnitWrite to floppy disk, can take hundreds of milliseconds to complete, so the interrupt latency (the time from occurrence of a physical interrupt to its being serviced) can be quite long on even the fastest of p-System hosts. When one is servicing a 9600 baud CRT, lapses of hundreds of milliseconds

simply won't do.

The 'solution' to this problem is to create a piece of software (usually in assembly language) called a Basic Input Output System (BIOS) to execute under the interpreter, buffer the interrupts, and provide I/O services. The 'hard' interrupts which the hardware generates are fielded by the BIOS and queued for processing by the p-System when the next p-code completes. In effect, the BIOS converts 'hard' interrupts into 'soft' interrupts. In the case of some devices, such as a CRT, the actual hardware services are provided by the BIOS, and the p-System just sees a buffer of characters from the device: the p-system does not in fact perform physical hardware device services at all.

When one Attaches an interrupt to a semaphore, the usual expectation is that a process will wait on that semaphore and serve as an interrupt service routine (ISR) for that interrupt. The question which arises at this point is whether one desires the Attached interrupt to be a hard one or a soft one. Another way to put this is to ask whether we want an interrupt when a character moves between the p-System and the BIOS or between the BIOS and the hardware?

Consider the following. Normally, when one gets an interrupt from an output device, such as a UART serving a CRT, that interrupt signifies the completion of the previous output operation and the availability of the device for another operation. This is the hard interrupt, and if it is sent in to the p-System a bit of confusion may result. As described above, the p-System code sees the CRT as being at the end of a long buffer of characters, and the fact that a physical output event has completed is of no interest to it. What the p-System code is interested in is when the output buffer has enough room in it that a "write" to that buffer will be successful without hanging up. Clearly, the p-System programmer desires soft interrupts when s/he performs an Attach.

In a similar fashion, on input, the p-System programmer is interested in knowing when a character is available in the input buffer, not when the key is struck. There is no purpose served in signalling a semaphore furiously as keys are struck: if the program was unable to respond to the initial signal, it will still be unavailable. What is needed instead is for a signal to be created whenever 1) a character enters an empty input buffer from the keyboard, and 2) a character is removed from a non-empty input buffer.

This seems natural enough, but some current BIOS implementations have provided soft [buffered] I/O to the p-System programmer while at the same time providing hard interrupts to the Attach facility. The point seems clear: p-System BIOSes should be specified to generate soft interrupts, or rather, to generate interrupts as data transitions between the p-System and the BIOS, not between the BIOS and the hardware.

ASE Macros

By: **Eric Eldred**

Here are two ASE macros you guys asked for, to place page end markers in your text a la WordStar, assuming you have a formatter such as ITP which allows a non-printing line in your text.

```
[1]
   |{ Take this up into say <f1> or else make }
   |{ into autotakeup }
|"UnPaginate"|s~
   |{ ASE 0.9r; uses <f7>, tag; by Eric Eldred 30 Jan 84 }
   |{ deletes page end markers after each 59 lines }
   |{ do this before Paginate }
st see |f7 / \comment~~~page~end|n/ |e
   |{ replace with any non-printing line}
   |{ or you could even put in a command to force }
   |{ a page break }
~ jf i |n |e |u b aL |e i |f7 |e d |n |e jf /r7// |.
```

{ Chaircritter Note: For ASE Versions 0.8x and before, change 'see' in the second executable line to 'seu', and the two instances of 'jf' in the third to 'ns'. The BackSlash (\) is the 'ignore this line' character in Jim Gagne's ITP text formatter -- other formatters, such as Sprinter, will use other characters for this purpose -- check your documentation }

```
[2]
   |{ Take this up into say <f2> or else make into }
   |{ autotakeup }
|"Paginate"|s~
   |{ ASE 0.9r; marker in <f7>; by Eric Eldred 30 Jan 84 }
   |{ sets non-printing page end markers after 59 lines }
   |{ UnPaginate first }
   |{hit <f2> for each page end; gives you a chance to }
   |{ fix widows }
59 |d i |f7 |e
   |{ when finished, J{ump to T{ag and resume editing } |.
```

Actually [1] is very tricky, as it includes a <return> character as part of the <f7> macro. When I tried to do a global replace of <return> characters by say "<return> 3" in 0.9q or even earlier UCSD editors, I managed to wipe out my files, occasioning Arley's [when he was still with Volition] sending me ASE 0.9r, which won't allow R(eplacing CTRL-Ms -- except by means of the cleverly placed backdoor exemplified by "/r7//".

{ Chaircritter Note: I learn something every day -- the "/r7//" means:
"/" -- Repetition factor -- effectively 'infinity'
"r" -- Replace
"7" -- the contents of Function Key #7 (|f7) -- I didn't know you could do this! It's not in *my* documentation!
"//" -- with nothing!

Effectively, "Search the whole file, and every time you find whatever is in 'f7', delete it." Pretty slick!

A better way to do this, I think, would be to define a '$PROFILE' macro as follows:

```
|x|*|f7|*|f2|*|f1|.
|'Page End' \Comment -- Page End|n|.
|'Paginate'|s~59 |d i |f7 |e |.
|'UnPaginate'|s~ st ne i |f7 |e ns /r 7 // jt|.
```

You put the cursor on the '|' before the 'x', press S(et M(arker, type in "$PROFILE" (without the ""s) and hit <Return>. Whenever you edit the file, ASE will jump to the macros and take them up, and they'll be ready to use any time you want them.

The first line says: "Take this line up in f1, and execute it -- take up the next line, put it in f7, take up the one after that, and put it in f2, and the next in f1."

The second line is the non-printing string to load in f7.

The third (in f2) goes down 59 lines and inserts the contents of f7, and sends an ETX (Control-C or equivalent).

The fourth line S(ets the T(ag wherever the cursor is sitting, puts a copy of the contents of f7 at the end of the file (so it can find at least one copy of it when it is deleting), jumps to the beginning of the file, deletes "f7" wherever it finds it, and jumps back to where you started.

The only problem I can see with this is that if you have non- printing lines or printing commands in the file, they take up a line on the screen, but *not* in the printed version, and the non-printing comment line will need to be adjusted accordingly to reflect the *actual* end of the *printed* page... }

Benchmarks on Turbo Pascal

By: Stephen F.B. Pickett

Alright, you wanted it! So you got it! I have run the famous Bondy benchmark on Turbo Pascal and a variety of p-code systems. The results were all obtained on a TIPC, running at the same clock frequency as an IBMPC, so the results may be regarded as interchangeable with the IBM results previously reported by myself and Jon Bondy. I have therefore restricted myself to highlights, in order to illustrate the points.

1. As expected, Turbo is *fast*.

2. All times for 32000 loops, in seconds. I discovered afterwards that these times (except for Turbo which I did by hand as there is no "time" function) are "slower" than the IBM's, tho' I am told that the TI runs a little faster than the IBM. The discrepancy comes, I believe, from the real time clock, which on the TI really does tick exactly every 100 milliseconds. On the IBMPC it ticks every 55.5 but I corrected it as it it were every 50 msecs. Unfortunately, all the programs were ready to run for the IBM and I couldn't find the source. So there is a built in 10% margin of error or fudge factor!

3. Turbo uses 3-word reals. My p-code results are for 4-word reals. 2-word reals are about 2 to 2.5 times as fast as 4-word reals. Turbo doesn't have 8087 support yet(it's promised), but they may have difficulty implementing 3-word reals! (subtle technical joke)

Bench Number & name	IV.1 p-code	Turbo	IV.1 8087	IV.1 NCG	IV.1 87NCG	Notes
1. For Loops	5.72	1.00		0.98		
(All times below have loop subtracted out)						
6. Integer Add	3.19	0.40		0.66		
7. Integer Mul	4.40	1.21		1.76		
8. Integer Div	4.62	1.46		1.87		
10 Real Add	28.58	32.41	15.60		2.75	e)
11. Real Mul	44.30	80.74	15.72		2.75	
12. Real Div	77.93	140.90	16.38		3.52	
13. Integer Xfers	1.87	0.23		0.55		
14. Int arr Xfers	15.06	1.06		11.87		a)
17. Int rec Xfers	3.41	0.38		0.55		
18. Rea rec Xfers	7.69	2.80	7.80	8.79	1.54	b)
19. Int IF	2.86	0.35		0.44		
21. Case	5.06	0.77		1.32		
22. Procedures	10.33	2.20				c)
26. Set Unions	28.03	25.46				c)
29. Pointer Xfers	4.07	0.74		0.66		d)

Key to Notes:

a. Bill Bonham has already shown the p-System's inferior performance can be greatly increased in all cases by running (*$R-*). Turbo runs without range-checking unless you specify, which I didn't since the original benchmark didn't specify either way!

b. Hooray, we actually won one, by cheating in 2 different ways!!

c. Everybody knows that sets and procedure calls don't native-code well!

d. p-System wins without cheating.

e. Interesting conclusion - in an interpretive situation, the best a floating point chip will do for you is to speed * and / up to the same speed as + and -. I have observed this to be also true for Z80/AMD9512.

The code file BENCH.COM was 30 blocks long. BENCH.CODE was 15. The results more or less speak for themselves. I should fill in with some impressions of using Turbo, however limited they may be. The preparation of the benchmark was easy. I had to do three things to my text file once it was T(ransferred onto an MSDOS diskette:

1. Make all num_loops and numloops the same. Turbo distinguishes.
2. Take out all references to RepFile. I couldn't be bothered to learn the syntax to Assign, which is much more like MS-Pascal than UCSD reset/rewrite
3. Give a string a size. There are no default string sizes in Turbo

It must be pointed out, in all fairness, that the interactiveness (as defined by Barry Demchak) on compiler errors is amazing, since you hit <esc> and find yourself in less than a second sitting in a Wordstar look-alike with the cursor at the right(sic) spot. Also Borland have solved in one the ANSI cursor positioning problem, by defining a single parameterisable sequence to handle EVERY terminal known to man including ANSI. If only SMS had been smart enough to do that!

Except for the above changes, it was so easy to compile and run that I had an executable MSDOS COM file almost before I knew it had started compiling. You can run the program in memory, without leaving the Turbo environment. This is quicker but you cannot then save a .COM file, a (quick) recompile is necessary. From the bench marks, and the relatively simple program environment (I didn't say it was good, just simple) you can guess how quick the (one-pass) compilation was. It hurts me to quote the statistics.

There is a very crude assembly language interface supplied with the documentation, specifying the parameter passing conventions, and it seems moderately obvious that these are in fact the ones Turbo uses itself. They even have a trick way to handle interrupts in PASCAL, though it's rather dumb and involves dropping the machine code to save the registers INLINE inside the pascal routine as a set of HEX digits. ABSOLUTELY NOTHING is written on how to LINK (or even ASSEMBLE) machine language routines. I am afraid this could turn into just another hacker's paradise as a result, since transportability is no way assured at any such interface.

I leave the rest in tabular form, perhaps there will be expanded entries or more of them when someone delves further into a neat little system. If only we could have started with this 4 years ago! Then perhaps all the present nice features could have been added without speed cost? Certainly it's worth bearing in mind if anyone ever redesigns a IV.x upwards-compatible p-machine. No, please don't tell me about II.x, it runs much slower than IV.x on the IBM, and if "tweaked" would only be marginally faster.

TURBO PASCAL

For:	Against:
Speed	Lousy real support
COM files	No UNITS, therefore NO LARGE PROGRAMS (yet)
	(they promise 'overlays')
Cheap	No PACKED arrays(packing occurs, but at random)
Semi-compatible	No Dispose
	No Get and Put, untyped files are 128 bytes not 512
	Underscore is significant in names
	Editor is slow, ASE badly needed
	Assembly language is undocumented
	Larger code files

P-SYSTEM

For:	Against:
Small code files	Slower running
More flexibility	Larger overheads (O/S etc)

```
More portable       Cannot run stand-alone
De facto standard   Developments hindered by backwards compatibility issues
8087 in interp      Expensive (finally becoming less so)
Real support
```

But combined with the DOS-Files p-System I have (and will be releasing shortly, I hope), it would be very tempting to do all quickie programming under TURBO. Except for the ASE problem. I know people who use the p-System *purely* for ASE's convenience, and everything else under M*-**S. It really is dreadful to have to go back to W******r with its hokey little control characters just after I finally got to use *all* my extra keys up! Many people who have tried using MS-PASCAL will think this is so fantastic, though, that someone who could offer an ASE-like editor to run as a TURBO program would be permanently and gainfully employed. Who knows, perhaps I'll even do it myself!?

Product Announcements

MACINTOSH DEVELOPMENT SOFTWARE AVAILABLE FROM SOFTECH MICROSYSTEMS

San Diego, CA, August 6, 1984 -- SofTech Microsystems announced today that they will begin shipping a full software development environment for the 128K Macintosh microcomputer on August 22, 1984.

This announcement makes available the first compiled Pascal and FORTRAN languages on the Macintosh. The new release includes a complete development environment with UCSD Pascal , FORTRAN-77 and an advanced development tool kit with the only 68000 Assembler available for the Mac.

"We have been extremely enthusiastic about the Macintosh since its introduction and knew that our extensive development environment would be ideal for the product," said Larry Allman, vice president of marketing. "Our system provides not only a vehicle to write large and complex applications on the 128K Mac but paves the way for a host of UCSD Pascal applications to be made available on the Macintosh very quickly," Allman continued. The software packages offered include:

- UCSD Pascal Development System - allows access to mouse, graphics and text fonts provided by Mac ROM routines, thereby allowing the application to take advantage of a popular set of Mac user interfaces for $195.00.
- FORTRAN-77 Development System - provides the user with an ANSI-77 subset FORTRAN with support for structured programming and improved character types for $295.00.
- Advanced Development Tool Kit - augments the software development environment provided with UCSD Pascal and/or FORTRAN-77. The tool kit includes source code for graphics/mouse interface, a symbolic debugger, 68000 Assembler, and a linker. The advanced development tool kit allows the developer to analyze and optimize programs during development. The cost is $150.00.

"SofTech's strategy enables the company to take advantage of emerging market trends as they occur," said Joe Taglia, product marketing manager for system software. "We're filling our obligation to the Mac world by developing the software needed," Taglia continued.

UCSD Pascal and Apple Pascal are sufficiently compatible so as to allow easy porting of Apple II and III programs to the Macintosh. In addition, UCSD Pascal applications running on other machine types such as the IBM PC can also be easily ported.

SofTech Microsystems, a subsidiary of SofTech, Inc. in Waltham, Massachusetts, is a leading supplier of microcomputer software products. It's major products are UCSD Pascal, the de facto standard Pascal for microcomputers, and the p-System , a popular microcomputer operating system that runs on machines from all major manufacturers including Apple, Hewlett Packard, DEC, IBM, Sage, Texas Instruments, and Zenith.

UCSD Pascal is a registered trademark of the Regents of the University of California. IBM is a registered trademark of International Business Machine Corporation. p-System is a trademark of SofTech Microsystems, Inc. Apple Pascal is a trademark of Apple Computer, Inc. Macintosh is a trademark licensed to Apple Computer, Inc.

For further information please contact SofTech Microsystems Customer Sales Department at (619)451-1230.

Classified Ads

For Sale: Sage II with 512K RAM, dual 640K floppies, Freedom 100 terminal, and complete cabling. Available with complete development software for $2500, or with Timberline business software for $3000. Also, Sage IV with 18MB hard disk for $3500 (development) or $4000 (business). Contact Timberline Systems (503)684-3660.

USUS Software Library

From Jon Bondy

Anthony Pompa has not returned letters, and has been dropped as a distributor. We no longer have a Victor distributor...

Henry Baumgarten is now distributing only Sage and 8" formats; a new distributor, Dave McFarling, is taking over the IBM, NCI, Apple, and Softech Universal Medium formats. Thanks for the help, guys!

In addition, Jim Harvison is now distributing Sage and Apple formats as a "USUS" distributor, not as an "individual". When ordering from Jim, please make your checks out to USUS, not to Jim personally.

Prices have been standardized for all of those formats which are distributed by more than one person. IBM, NCI, and Sage prices have been set at $6 (1 volume/disk), $7 (2 volumes/disk), or $8 (3 volumes/disk 10-sector) (all 80 track). 8" disk prices have been set at $5 each. Apple][prices are set at $8.00 for a two disk volume.

If you have problems with a distributor, first contact them directly to see if the problem is a misunderstanding or some "reasonable" delay (death in the family, computer hit by lightning, etc). It is reasonable to expect a distributor to send you disks within 4 weeks of an order: delays longer than that should cause you to investigate further. If you don't get satisfaction, please *write* to both them and myself, describing the circumstances.

- Jon Bondy

From Harry Baya

Some of you are doing it! Don't deny it. You would be fools not to do it. I do it myself. The time has come to to admit it, to tell the world what you did, and why.

Telling the truth isn't always easy, but sometimes it can be of value to others who face the same temptations you did. We know they are going to do it - so lets make it easier for them.

The above text will be stored for future use in a letter to the "Kinks" section of Penthouse together with some choice USUS comments on byte-sex and bit-fiddling. It's function here is to ask you users of the USUS library to share some of your hard earned wisdom with others .

Some of you have used the USUS program library. Your experience could be of value to others. What were you particularly pleased with? What difficult situations deep in the Adventure game of UCSD Pascal did the library help you to get through?

This note is intended to start a column called "LIBRARY USERS" in this newsletter. It's goal is "library consciousness raising" within the USUS community.

The initial idea came up in a discussion at my first Library Committee meeting, in Toronto. We agreed that we would like to find ways to make the program library more useful to members. A library users' column can serve that purpose.

So, I am asking for your participation. Take out a piece of paper and jot down the programs that you can recall using from the library.

In my case I have used or shared :

- remtalk
- adventure

- lisp
- combine

```
function enough_allready(level: integer): boolean;
  begin
  If (this is as far as you want to go
       in this exercise)
   then
     begin
     Please send your notes to Harry.
     enough_allready := true;

     case  level  of
       1: begin
           My sincere thanks;
           Actually I would like more!
           May you learn about recursion
             the hard way!
           if enough_allready(1)  then
             you will never get to here;
           end;
       2: even more thanks;
       3: can I buy you a drink?
       4: eternal obligation
       end;  {case}
  end
else
  begin   {keep on truckin!}
  enough_allready := false;
  if level = 4 then
  begin
    Please send me your notes anyway.
    you are a glutton for work!
    call me!
    exit(realworld); {don't worry, you were
                      way down the recursive
                      stack to start with}
    end;
  end;
end;  {enough_allready}
If not enough_allready(1)  then :
```

Ahhh, you are still with me. Ok, Mark those programs that you found particularly useful. If not enough_allready(2) then mark any others where you could make useful, interesting or entertaining comments.

In my case I found "remtalk" particularly useful and could make some comments about all four of the programs I have used. I tripped out of this process at level= 3 in the next paragraph.

Now, if not enough_alleady(3) then pick one of the programs and write out your comments and send them to me. I will include as much as possible in this column.

I would also appreciate any suggestions for other ways of making the library more useful. Here are a few starters.

- "The all-time library hit parade"
- "Drop the rod before you catch the bird, but don't stand directly under the bird"
- A periodically updated disk containing a data base program and data file that would help you find out what is available and what you need to know about it. (e.g. will it work on an Apple][?)

My addresses are :

Harry Baya
The Pumpkin People
565 Broadway, 2H
Hastings-on-Hudson, NY, 10706

Telephone : (914) 478-4241
Compuserve: 72135, 1667
Telemail : HBaya/USUS

In closing, I will be delighted to get a lot of response to this. My feelings will be hurt if I don't get at least one reply. If you don't care about helping nameless masses, then help one needy individual.

```
If enough_allready(4) then
 if you sent something then goody!
 else
   they also serve who read this far.
```

Good luck! - Harry Baya

USUS Software Library Distributors

Distributor	Disk Format(s)/Prices
Henry Baumgarten 3325 Hillside Street Lincoln, NE 68506 402-489-6441 (h) 402-472-3301 (w)	Standard Sage and 8" prices; see above
Jon Bondy Box 148 Ardmore, PA 19003 215-642-1057 (h)	Standard Sage and 8" prices; see above Subtract $4/disk if you send him properly formatted and zeroed Sage disks. Subtract $3/disk if you send him formatted SSSD 8" disks
Clark Gestring 4643 W Oberlin Place Denver, CO, 80236 303-797-6739 (h) 303-694-8797 (w)	TI 99/4A SS SD 35 TK @ $13/4 disk volume. SS SD 40 TK @ $11/3 disk volume. DS SD 35 TK @ $10/2 disk volume. DS SD 40 TK @ $10/2 disk volume.
Kenneth K. Kam P.O. Box 3112 Torrance, Ca. 90510	Heath H-89 5-1/4 inch disks for $17/tri-disk vol.

Distributor	Disk Format(s)/Prices
Dick Karpinski 6521 Raymond Oakland, CA, 94609 415-666-4529	Northstar disks for $15/dual single-sided disk volume.
Jim Harvison Box 3277 Silver Spring, MD 20901 301-593-2994	Standard Sage and Apple prices; see above
David McFarling 3815 Adams Street Lincoln, NE 68504 Rockville Md, 20850 402-467-3591	Standard IBM and NCI disk prices (specify format and number of blocks); Apple // disks at standard prices; SofTech Universal Medium disks at $8/vol. (two disks)
George Schreyer Box 1645 Redondo Beach, CA 90278 213-371-0198	8-inch disks for $10/volume.
Marc Wigan Wigan Associates Box 281 Mt Waverley Victoria 3149 AUSTRALIA	8-inch disks and Sage disks. Pricing unknown.

Who Supplies What:

"Std" 8"	Henry Baumgarten, Jon Bondy, George Schreyer, Marc Wigan
Apple	Jim Harvison, David McFarling
Heath-89	Ken Kam
TI	Clark Gestring
IBM	David McFarling
NCI	David McFarling
Sage	Henry Baumgarten, Jim Harvison, Jon Bondy, Marc Wigan
NorthStar DD	Dick Karpinski
Victor 9000	Nil
CSI 6809	Nil
OSI	Nil

New USUS Software Library Volumes

USUS Library Volume 29
A script driven communications package and a weaver's helper
(a USUS REMUNIT is needed for CONVERS, see volume #15)

VOL29:

DRAW4A.TEXT	32	A simple pattern weave analyzer
DRAW4A.1.TEXT	34	an include file
DRAW8A.TEXT	32	A more complex pattern weave analyzer
DRAW8A.1.TEXTan include file		
DRAWDN.DOC.TEXT	26	Documentation for the weaversdesignpackage'
OSMISC_II0.TEXT	12	Misc routines for CONVERS for version II.0
OSMISC_IV.TEXT	14	Same for IV.x
TEXTIO_II0.TEXT	26	Text file routines for CONVERS for version II.0
TEXTIO_Iv.!EXT	14	Same for IV.x
SCRNOP_II0.TEXT	14	An ersatz SCREENOPS for II.0
CONV_TEST.TEXT	6	A test script
CONVDOC.TEXT	70	Documentation for CONVERS
INSTALL.TEXT	26	Installation notes for CONVERS
CONVERS.TEXT	112	CONVERS itself
TERMINAL.TEXT	8	A dumb terminal emulator which can stand alone
VOL29.DOC.TEXT	6	You're reading it

This volume was assembled by George Schreyer from material collected by the Library committee.

USUS Library Volume 30
Softech Microsystems' Performance Monitoring Tools, Part I

VOL30:

CLOCK.AST.TEXT	10	
CMDCODES.TEXT	4	
SAMPLE.TEXT	28	Time Profiler utility program
CONCURRENT.TEXT	12	Time Profiler replacement for IV.13 CONCURRE unit
PMINSERT.TEXT	22	Utility for inserting calls to Routine Monitor
DISASM.TEXT	16	
PMA.SAGE.TEXT	8	Time Profiler PMATTACH unit for SAGE II (For IV.2)
DISPLAY.TEXT	8	Displays contf ID translation file
PMT.AST.H.TEXT	8	
EXAMPLE.TEXT	4	
FAULTCOUNT.TEXT	28	Utility for cummulative fault history analysis
PMATTACH.TEXT	6	Time Profiler PMATTACH unit for IBM PC. (For IV.2)
FLIPDICT.TEXT	4	
PMS.IV1.TEXT	16	IV.13 version of Time Profiler sampling task
FLIPSEG.TEXT	6	
PMTIMER.TEXT	10	Timer unit using p-System 60 Hz Clock
MONITOR.TEXT	28	PERFOPS Start, End, Interact utility

OPTCONST.TEXT	10	
PMA.PC.IV1.TEXT	6	Time Profiler PMATTACH unit for IB PC (For IV.1)
OPTFILE.TEXT	32	
PMA.SG.IV1.TEXT	8	Time Profiler PMATTACH unit for SAGE II (For IV.1)
OPTPROCS.TEXT	44	
PMDISPLAY.TEXT	18	Remote fault analyzer display program
OPTREFS.TEXT	10	
PMPROBE.TEXT	8	Routine Monitor "probe" unit
OPTSEGS.TEXT	16	
PMSAMPLE.TEXT	16	Code for Time Profiler sampling task (For IV.2)
OPTUTILS.TEXT	16	
PMT.SAGE.TEXT	8	Timer unit using SAGE II clock
PCODES.TEXT	10	
REASM.TEXT	8	
VOL30.DOC.TEXT	12	You're reading it

This volume was contributed to the USUS Library by SofTech Microsystems

USUS Library Volume 31

Softech Microsystems' Performance Monitoring Tools, Part II

VOL31:		
PERFOPS.TEXT	6	
PERFOPS.A.TEXT	54	
PERFOPS.B.TEXT	36	
IV1.PSCLIO.CODE	11	
IV1.REAL4.CODE	12	
IV2.CONCUR.CODE	7	
IV2.PSCLIO.CODE	9	
IV2.REAL4.CODE	13	
CONCURRENT.CODE	3	Time Profiler replacement for IV.13 CONCURRE unit
DISPLAY.CODE	3	Displays contents of ID translation file
FAULTCOUNT.CODE	7	Utility for cummulative fault history analysis
MONITOR.CODE	10	PERFOPS Start, End, Interact utility
PERFOP.IV1.CODE	33	PERFOPS unit for use with a IV.1 p-System
PERPS.CODE	33	Performance monitor unit. (IV.2/Liaison)
PMA.PC.IV1.CODE	3	Time Profiler PMATTACH unit for IBM PC (For IV.1)
PMA.SAGE.CODE	3	Time Profiler PMATTACH unit for SAGE II (For IV.2)
PMA.SG.IV1.CODE	3	Time Profiler PMATTACH unit for SAGE II (For IV.1)
PMATTACH.CODE	3	Time Profiler PMATTACH unit for IBM PC (For IV.2)
PMDISPLAY.CODE	4	Remote fault analyzer display program
PMINSERT.CODE	36	Utility for inserting calls to Routine Monitor
PMPROBE.CODE	3	Routinitor "probe" unit
PMS.IV1.CODE	6	IV.13 version of Time Profiler sampling task
PMSAMPLE.CODE	6	Code for Time Profiler sampling task (For IV.2)
PMT.AST.CODE	3	Timer unit using clock on IBM PC AST board
PMT.SAGE.CODE	3	Timer unit using SAGE II clock
PMTIMER.CODE	3	Timer unit using p-System 60 Hz Clock
SAMPLE.CODE	10	Time Profiler utility program
VOL31.DOC.TEXT	12	You're reading it

This volume was contributed to the USUS Library by SofTech Microsystems

USUS Library Volume 32

Softech Microsystems' Performance Monitoring Tools, Part III

VOL32:		
TOOL.DOC.A.TEXT	38	Documentation files which describe how to use
TOOL.DOC.B.TEXT	??	the monitoring tools above. Printable using
TOOL.DOC.C.TEXT	22	PRINT.CODE, using the instructions in README.TEXT
TOOL.DOC.D.TEXT	28	The resulting document is about 60 pages long, so you
TOOL.DOC.E.TEXT	40	might write it to a file and spool it if you dont
TOOL.DOC.F.TEXT	32	want your computer tied up printing for a while...
TOOL.DOC.TEXT	4	
PRINT.CODE	19	utility to print the documentation
README.TEXT	4	instructions on use of the print utility
PDOSTRANS.TEXT	36	Utility from Mike Berg which allows MS-DOS files to
PDOSOPS.TEXT	52	... be read/written on PC or Sage or...
PDOSOPS.CODE	20	
PDOSTRANS.CODE	14	
LIFE.TEXT	28	Jai Khalsa's version of the Game of Life as ...
LIFE.DOC.TEXT	36	... discussed on MUSUS and TeleMail
LIFE.CODE	7	
VOL32.DOC.TEXT	12	You're reading it

This volume was assembled by Jon Bondy from material collected by the Library committee.

UCSD

Take P-codes in your Stride

Optimiser: *polishes P-codes. Smaller faster programmes. Works at code file level, access to source not required. Maintains portability. Can insert instruction to native code procedure. Can reduce code file size by 10-20% (Pascal), 30-50% (Fortran).*

Analyser: *provides information to improve programme performance. Tells you what your software is really up to. Variable and procedure usage. Case table overheads. Uncalled procedures. Performance monitor source.*

Edip: *P-code editor. Can edit code files direct, access to source not required.*

Demonstration disc $15, contains excerpts from manual, example programmes and demonstration Optimiser.

Eliacomputer, *1510 East 4th Street, Brooklyn, New York 11230, USA.*

Call (718)336-4834 or 4109.

Poptyser™

Poptyser is produced by Knowledge Software Ltd, 32 Cove Road, Farnborough, Hants GU14 0EN, England. (0252) 520667. (TMAIL KNOWLEDGE).

Poptyser is a trade mark of Knowledge Software.

USUS Software Library Order Form

Use this form to order the USUS Library volumes of your choice. Send the order DIRECTLY to the distributor, and make your check out to him/her, NOT to USUS.

Thirty-four volumes are now available. All 8080-specific code and CRT terminal data on Volumes 1, 2A, and 2B have been removed to form a single Volume 1 for the Apple. Volume 2B is normally shipped in CP/M format and has little utility for anyone without CP/M. Volume 21 is Apple-specific. A special Western Digital (WD) format disk contains a Mapper program, allowing owners of WD machines to read DEC-format disks, the USUS standard 8-inch format. WD owners CANNOT obtain double-density disks from USUS; ask WD for help converting single to double density on your machine. DEC format begins on track 1/sector 1, with interleave 2 and skew 6.

NAME, TITLE _____

COMPANY (if work address) _____

NUMBER AND STREET _____

CITY, STATE, ZIP, COUNTRY _____

SYSTEM (PLEASE fill in) _____

DISKETTE FORMAT REQUIRED (circle one):

Standard 8-inch SSSD NorthStar Apple OSI

Heath H-89 IBM PC IBM PC TI SAGE

VOLUMES REQUESTED (circle one or more):

1 2A 2B 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 32 UK3 UK4

Price per volume (see distributor list) $_____

Total for disks $_____

Sales tax (if intra-state sale) $_____

TOTAL AMOUNT ENCLOSED ($U.S.)$_____

(NOTE: ALL orders MUST be prepaid!)

IMPORTANT: PLEASE READ AND SIGN THE AGREEMENT ON THE FOLLOWING PAGE, OR YOUR ORDER CANNOT BE PROCESSED!

Software Order Agreement

I will not permit the Programs I receive pursuant to the foregoing order to be published for profit in whole or in part or to be transferred to any person who is not a member of USUS, without the express written consent of the author identified in the files of the Library. Further, lacking such author's consent, I will ensure that the following items have been submitted to the designated individual (for now the USUS Treasurer, USUS Library Chairman, or a Library Distribution Subcommittee Chairman) before transferring any Program(s) to another USUS member: a) the current Software Order Agreement signed by the software recipient; and b) $1 per volume or fraction of a volume received.

I acknowledge that neither USUS nor any of its representatives nor the author makes any warranty with respect to the Program, particularly NO WARRANTY OF FITNESS FOR ANY PURPOSE, and that the Program may require extensive alteration by an expert in programming before it will suit my needs.

(Signed) _____ Date _____

The Stride 400 Series

From $2900 to $60,000+

As you can see below, the Stride 400 Series has no shortage of features. We've recently added RAM expansion to 12M bytes, low-cost high-speed graphics and a revolutionary hands-free NOD™ cursor control device. But there's more to Stride Micro than hardware. We're an open company, sharing source programs and schematics with our users. That's the reason Stride and Sage (our former name) machines are preferred by many leading p-System developers. It's what we call "Performance by Design".™

Stride 400 Series Technical Specifications

- *10 MHz 68000 CPU (12 MHz optional)*
- *VMEbus interface*
- *256K bytes of parity RAM standard Up to 12M bytes with no wait-states*
- *5¼" 640K byte floppy disk drives*
- *10M to 448M bytes hard disk storage*
- *Battery backed-up real-time clock*
- *4K bytes of battery backed-up CMOS RAM*
- *4 to 22 RS-232C serial ports*
- *Omninet networking hardware*
- *p-System IV.2 w/LAN software*
- *High speed low-cost monochrome graphics (784 x 325 resolution)*
- *NOD cursor control device*
- *FPU hardware floating point*
- *Streaming ¼" tape drive backup*
- *MMU Memory management unit*
- *Stride configuration utilities*
- *TeleTalker communications software*
- *Centronics compatible parallel port*

The Stride Multiuser p-SYSTEM

Stride provides a complete multiuser system **standard** with every machine. This unique system allows multiple p-System (and CP/M-68K) operating systems to co-exist at the same time on a single machine. An easy-to-use beginner's program (MU.BUILD) will generate a multiuser system within minutes. Sophisticated users will appreciate the highly powerful and flexible MU.UTIL program which allows an incredible level of customization of the multiuser system. It is a multitasking, propriety BIOS developed in 1982 (the first commercial multiuser p-System), and has grown in strength and flexibility ever since.

STRIDE MICRO

Stride Micro, 4905 Energy Way, Reno, NV 89502 (702) 322-6868, TWX 910-395-6073.

USUS ™
P.O.Box 1148
La Jolla, CA 92038

Bulk Rate
U.S. Postage
PAID
Brooklyn, NY
Permit No. 316