Pascal News
Communications about the Programming Language Pascal by Pascalers

- APL Scanner
- Computer Generated Population Pyramids
- Path Pascal
- Introduction to Modula-2
- Validation Suite Reports
- Announcements
POLICY: PASCAL NEWS

Pascal News is the official but informal publication of the User's Group.

Purpose: The Pascal User's Group (PUG) promotes the use of the programming language Pascal as well as the ideas behind Pascal through the vehicle of Pascal News. PUG is intentionally designed to be non-political, and as such, it is not an "entity" which takes stands on issues or support causes or other efforts however well-intentioned. Informality is our guiding principle; there are no officers or meetings of PUG.

The increasing availability of Pascal makes it a viable alternative for software production and justifies its further use. We all strive to make using Pascal a respectable activity.

Membership: Anyone can join PUG, particularly the Pascal user, teacher, maintainer, implementor, distributor, or just plain fan. Memberships from libraries are also encouraged. See the COUPON for details.

Pascal News is produced 4 times during a year; January, April, July October.

ALL THE NEWS THAT'S FIT, WE PRINT. Please send material (brevity is a virtue) for Pascal News single-spaced and camera-ready (use dark ribbon and 15.5 cm lines!)

Remember: ALL LETTERS TO US WILL BE PRINTED UNLESS THEY CONTAIN A REQUEST TO THE CONTRARY.

Pascal News is divided into flexible sections:

POLICY — explains the way we do things (ALL-PURPOSE COUPON, etc.)

EDITOR'S CONTRIBUTION — passes along the opinion and point of view of the editor together with changes in the mechanics of PUG operation, etc.

APPLICATIONS — presents and documents source programs written in Pascal for various algorithms, and software tools for a Pascal environment; news of significant applications programs. Also critiques regarding program/algorithm certification, performance, standards conformance, style, output convenience, and general design.

ARTICLES — contains formal, submitted contributions (such as Pascal philosophy, use of Pascal as a teaching tool, use of Pascal at different computer installations, how to promote Pascal, etc.).

OPEN FORUM FOR MEMBERS — contains short, informal correspondence among members which is of interest to the readership of Pascal News.

IMPLEMENTATION NOTES — reports news of Pascal implementations: contacts for maintainers, implementors, distributors, and documentors of various implementations as well as where to send bug reports. Qualitative and quantitative descriptions and comparisons of various implementations are publicized. Sections contain information about Portable Pascals, Pascal Variants, Feature-Implementation Notes, and Machine-Dependent Implementations.

VALIDATION SUITE REPORTS — reports performance of various compilers against standard Pascal ISO 7185.
Pascal News
Communications about the Programming Language Pascal by Pascalers

JULY 1983    NUMBER 26

2 EDITOR'S NOTES
5 OPEN FORUM
SOFTWARE TOOLS
11 Program APL Scanner
   By Vincent Dichristofano, Alan Kaniss, Thomas Robinson and John Santini
ARTICLES
26 "Don't Fail Me Now" By Sruly Blotnick
27 Computer Generated Population Pyramids Using Pascal By Gerald R. Pitzl
32 Path Pascal — A Language for Concurrent Algorithms By W. Joseph Berman
37 An Introduction to Modula-2 for Pascal Programmers
   By Lee Jacobson and Bebo White
BOOK REPORT
41 Data Structures Using Pascal
ANNOUNCEMENTS
42 SBB Announces Pascal Compiler for IBM PC
42 Sage Opens Boston Division
42 New 16 Bit Sage IV
43 New Modula-2 Manual
44 USUS Fall Meeting
44 Text Editor Interest Group
44 Modula-2 Users Group
45 USUS San Diego Meeting
46 Volitions Modula-2 for IBM PC
IMPLEMENTATION REPORT
47 VALIDATION SUITE REPORT
48 OmegaSoft Pascal Version 2
SUBSCRIPTION COUPON
VALIDATION SUITE COUPON
USUS MEMBERSHIP COUPON
Hello,

Well, this is the third issue I am involved with and there have been many changes. I would like to write of Pascal first.

Pascal has enjoyed a jump in attention in the last year. One reason is that there are Pascal compilers available for many machines and, I am tempted to say, they are available for any machine. Most of the major main frames have Pascal either directly or from a third party.

One step down in size, I know of only one machine, the Tandem computer which is without a Pascal implementation. A Tandem representative here in Cleveland informed me they have a language called "TAL" and in many cases will execute a Pascal program with no changes.

A couple more steps down in size are the small Digital Equipment machines and compilers are available from about four sources. IBM has the Display writer and Datamaster. These were released without our language, but in the last year, UCSD Pascal has been made available through IBM. Apple Computer has been a strong and long supporter of Pascal. TRS 80 has UCSD Pascal.

The smallest machine with Pascal is the TI 99/4A. In this size, Commodore has promised Pascal for this summer on the "64" and "128" machines.

The small computer, that is, the home computers and small business computers, have exceeded $10 million in sales. This is according to Future Computing, a Richardson, Texas research firm.

With a guess, I would say that Pascal is implemented on at least 25% of these machines. If only 1% of these were being used to learn and program Pascal, then 25,000 people are presently involved. This is a lot of people looking for the best books from which to learn.

I am making an appeal to our members to submit comments and reviews of text books so that we all may benefit from your experience. I get calls from authors requesting information on Pascal. To these people, the best I can do is to send complete sets of Pascal News! With your comments and criticism, perhaps we could influence future text books.

Herb Rubenstein of Budget Computer in Golden, Colorado has sent a small article from Popular Computing. It seems that advanced placement test in computer science will use structure programming and the Pascal language. These tests allow up to one year of college level credits in computer science. The author of this article, Dan Watt, believes that the choice of Pascal in the testing may lead to Pascal as a defacto standard in high schools preparing students for college. Let me quote the last paragraph:

"This situation illustrates the power of the testing establishment to influence the lives of students and teachers. Although the vast majority of high schools now offer Basic as the standard computer language for most programming and computer science classes, this action by the College Board may lead to the establishment of Pascal as a defacto standard for high school teaching and spawn an entire mini industry of curriculum to meet the new requirements. It may also offer significant school marketing advantages to microcomputer companies that already support Pascal - such as Apple, IBM and Texas Instruments."

I would like to see comments from you regarding this use of Pascal in a rite of passage.

In this issue, you will find a reprint of Dr. Srully Blotnick's column from Forbes magazine. I like this column because of the clever way he has made our economy dependent on you learning Pascal.

I enjoy Forbes magazine. They emphasize common sense and illustrate proven business practices. Forbes also takes a pulse of industries, and small computers is a fast growing industry. In a column called "Technology", edited by Stephen Kindel, on March 28, 1983, he noted that 2% of the households in the U.S.A. own computers of one form or another. There had been predictions of 40% of households by 1990. This has been reduced to 20% in 1990 because there doesn't seem to be software that is useful in households.

Mr. Kindel ends this article with a quote from Seymour Papert, an MIT professor: "The real purpose of learning how computers work should be to improve human logic and thought processes, to make people more creative, not simply more dependent on machines."

Maybe this would be a good issue to review the tools available in our back issues. This issue contains the APL scanner. I am embarrassed to print this, not because of the program's quality, but because it was submitted four years ago. Well, no time like the present.

In issue #17 (yellow), Arthur Sale submitted "Reference", a procedural cross reference. This program provides a printout of the heading of each procedure and function with indentation showing nesting. In issue #25, Mr. Yavner has improved on this program with "A Better Referencer". Mr. Yavner claims that Pascal News has been his sole source of instruction in Pascal. I believe this is a compliment to Andy Mickel and Rick Shaw for their efforts to maintain this newsletter. We should also thank our contributors, Mr. Sale for instance, for outstanding generosity. These people will appreciate your complements, criticisms and gifts of money. (Ho! Ho!)

Andrew Tandembaum, in issues 21 and 22/23, provided us with "The EM1 Compiler". This is a good look at all that is necessary for a pseudo 32-bit machine pascal compiler.

The UCSD Pascal Project started with a 16-bit pseudo machine portable compiler. It was called P4 out of Zurich, Switzerland by Vrs Ammann, Kesav Nori and Christian Jacobi. I mentioned this because it has been published with critical commentary by S. Pemberton and M.C. Daniels in 1982. It is presented as a
case study of compiler design and is very interesting to read.

**Pascal Implementation**
S. Pemberton and M.C. Daniels
Ellis Horwood Limited Publishers
Distributed by:
John Wiley & Sons
605 Third Avenue
NY, NY 10016
USA

In #21 you will find Jeff Pepper's fine implementation of extended precision arithmetic.

Nicklaus Wirth, Pascal's creator, wrote Pascal S and we have it in #19 (mislabeled #17). This is a subset of Pascal and was intended as a teaching aid.

Also in #19 is a Lisp interpreter written in Pascal.

“MAP”, a Pascal macro preprocessor for large program development, is published in #17.

Issue #16 contains the Validation Suite version 2.2. This is the compiler checker that Arthur Sale and Brian Wickman have now revised to version 3. This new version is available by using the Validation Suite coupon in the rear of this issue.

“Prose”, a text formatter, by John Strait is the major program available in #15. A disclaimer in the instructions manual admits that it doesn't do everything, but I must say, it has a lot of capability.

In #13, two programs were printed that performed the same work. A sort of “Battle of Algorithms”.

“Pretty Print” and “Format” used any Pascal programs as input and printed it in a consistent style.

For those of you looking for other Pascal periodicals, there are four of which I know. “Pascal Market News”, 30 Mowry Street, Mt. Carmel, CT. 06518. This is a nice quarterly for $9.

Another quarterly for Oregon Software users is the “Pascal Newsletter”. Maybe this is too narrow in content, but you will know what Oregon Software is up to.

Their address is 2340 SW Canyon Rd., Portland, OR. 97201.

A very slick magazine with good design is “Journal of Pascal and Ada.” You can contact them at West Publishing Company, 898 South State Street, Orem, UT. 84057. The cost is $14 for six issues.

The USUS News and Report is more a system user’s journal, but the system is based on Pascal. They also have a software library, seventeen floppy disks full, and all in source code and written in Pascal.

Now to the business of Pascal News. Pascal News, as the Pascal periodical granddaddy published since January 1974, has had its ups and downs. In 1979 our circulation was 7,000; now it is 3,600. Our biggest problem has been irregular publication. I am committed to four issues this year and I am considering six issues next year. I believe that regularity will supply us with growth and members and more software tools.

As I mentioned in the last issue, PUG (AUS) has stopped and I, in the USA, have taken over their area. Unfortunately, they have not sent me their mailing list and I fear that I have lost touch with our members there. This issue will be sent to those members listed as of 1979 and I hope they will “spread the word” and the subscription coupons!

Our PUG (EUR) has performed very nicely and I thank Helmut Weber and friends for their good work. But they have a problem concerning money. They have not charged enough for subscriptions and were pressed to send out #24. As a result, I will mail all issues directly and I hope you will not be inconvenienced. Please keep in touch with them as they are a strong group.

I have saved the worst for last. In November, 1982, I sent 300 copies of issue #24 to Nick Hughes in care of PUG (UK), Post Office Box 52, Pinnen, Middlesex HA5 3FE, United Kingdom. Using the phone number 866-3816, the air express shipper delivered these issues by mid-November. All well and good. The issues arrived before the cover date with plenty of time to post them to our English members. I called Nick at this number many times, but spoke to him only after many months. It was late April and I asked if I should use the same procedure in shipping #25 to him.

Nick said that the issues arrived properly and that method was efficient but wanted to know what was in #25. He told me that he did not like issue #24 and from the sound of it, did not like issue #25. He had disliked #24 so much, he decided not to send any of them out. Need I say more?

Nick will not supply his mailing list so I am sending this issue and #25 directly to the members of record in the United Kingdom as of 1979. If you feel a need to find out why Nick Hughes did not like issue #24 or would like to see it yourself, please call or write Nick at the above address and ask for your copy. He has 300 and I am sure he can spare one.

As a result of these difficulties, I will receive and service all subscriptions from here in Cleveland, Ohio. From now on, there will be only one person to blame if you have a complaint.

As of this issue, a year’s subscription is raised in price to $25 a year and $50 for three years. These represent two sets of costs: production and organization. Production costs are typesetting, printing and mailing. Other activities of production are editing, reviewing, quality assurance and formatting. These tasks are performed by "yours truly" and presently I do them for free. (I'm real smart!)

Organization is a cost of servicing you and other members satisfactorily. This includes collecting and reviewing the mail, depositing checks, updating the mailing list, sending back issues to fill new subscriptions and sending sets of previous years back issues. In order to do this correctly, and in a timely fashion, I don't do it. I pay a firm to perform "fulfillment" and it takes one or two days per week. This cost is small compared to the bad feelings generated if not done correctly and quickly.

These are costs of which you are totally responsible. This newsletter has been a beneficiary of volunteerism. There are no volunteers now (save me). In many magazines, advertisements will pay for all production and organizational costs plus provide profits, sometimes large profits.

The cost of a full page ad in Byte or PC or PC World is over $2,000 and these are publications with 500 pages!

Now we may be able to keep our costs down and publish more often if we accept advertising. Three hundred dollars per page is not expensive. I will pursue

Open Forum
advertisers and I am asking for your help. If you are writing a book, have your publisher advertise with *Pascal News*. If you are making software packages, influence your boss in the virtues of an ad in *Pascal News*. If you manufacture or sell computers, sell your product from the pages of *Pascal News*. This is the oldest Pascal publication and, I proudly say, the most influential.

This newsletter helps spread Pascal and our members were most influential in the standard efforts.

I believe *Pascal News*’ new mission is to enable Pascal to be taught in the easiest way. This is in many forms. For instance, reviews of books and texts, discussion of what features to teach first as a foundation, how to teach advanced courses, discussions of extensions or standard program tools to include in every well written program as it is appropriate.

By the way, Andy Mickel tells me that the ‘‘Pascal User’s Manual and Report’’ by Jensen and Wirth has sold 150,000 copies in 1982. This is interesting considering that in the previous seven years, it sold 175,000 copies. A very sharp jump in interest.

A new textbook has been sent to me, ‘‘Pascal’’ by Dale/Orshalik, 1983 DC Heath. A nice title, short and to the point. The preface states a philosophy that I would like you to comment on.

‘‘In the past there have been two distinct approaches used in introductory computer science texts. One approach focused on problem solving and algorithm design in the abstract, leaving the learning of a particular language to a supplemental manual or a subsequent course. The second approach focused on the syntax of a particular programming language, and assumed that the problem-solving skills would be learned later through practice.

We believe that neither approach is adequate. Problem solving is a skill that can and should be taught — but not in the abstract. Students must be exposed to the precision and detail required in actually implementing their algorithms in a real programming language. Because of its structured nature, Pascal provides an effective vehicle for combining these two approaches. This book teaches problem-solving heuristics, algorithm development using top-down design, and good programming style concurrently with the syntax and semantics of the Pascal language.’’


Two notes from members:

Steven Hull of Campbell, California, received a notice from me that #22/23 had been returned to us because the postal service will not forward bulk mail. His reply:

‘‘I guess this will teach me to move from Lakewood (a suburb of Cleveland, Ohio). Didn’t know bulk mail wasn’t forwardable. The Postal Diservice has been re-routing every piece of junk mail for a full year . . . I might have to file suit to stop it all!’’

And from Eric Eldred of New Hampshire who rewarded *Pascal News* with a three year subscription and dutifully filled the coupon with name and address and arrived at a request for ‘‘Date’’. Eric filled in ‘‘No! Married!’’. Thanks Eric, I needed that!

Charlie
To Charlie Gaffney,

I’m glad you have taken on *Pascal News*. I hope it works.

Perhaps, I should say what I would like to see published in *Pascal News*. The most valuable things are 1) Tools, and 2) Info on the various implementations. In my job we are using many computers. It is very helpful to know which compilers work well, meet standards, and produce efficient code. Apple Pascal is nearly bug free, and works as specified (with UCSD quirks). IBM Pascal VS is good — extensions are large presenting conversion problems if they are used. It has a good interface to FORTRAN. VAX Pascal is plain vanilla, appears to work well but we have not tested it in difficult situations. HP Pascal 1000 works fine but does not have a stack architecture and seems to compile slowly. Recent tests on HP Pascal 1.0 for the HP 200 computers seem to indicate it derives from UCSD although it is a native code 68000 compiler. It seems to work very well. We are interested in Pascal for the Data General Eclipse.

Good luck,
Dennis Ehn
215 Cypress Street
Newton Centre, MA 02159

Gentlemen:

Would you be so kind as to send information on the Pascal User’s Group (PUG) and its official publication *Pascal News*. Recently we have acquired a microcomputer Pascal compiler and are very much interested in keeping up with current developments in Pascal.

Our system is based upon a SouthWest Technical Products Corporation S/09 computer, running the UniFLEX Operating System (similar to UNIX). If specific information is available for this unit, please let us know.

Additionally, the college has several (approximately 18) Apple computers which are capable of running the UCSD Pascal System. Once again, any special information here would be very helpful.

We look forward to hearing from you and hope that we can make a positive contribution to the Pascal User’s Group.

Yours Truly,
Lawrence F. Strickland
Dept. of Engineering Technology
St. Petersburg Jr. College
P.O. Box 13489
St. Petersburg, FL 33733

December 1, 1982
I hope the letter referring to the possible end of the P.U.G. is wrong! I can be of some help if needed.

Allen Duberstein
Pine Instrument Co.
3345 Industrial Blvd.
Bethel Park, PA 15102

January 10, 1983
Dear Mr. Gaffney:

Enclosed is a check covering both the remailing cost of *Pascal News* #24 ($5) plus my membership renewal for two years ($18).

My apologies for getting out of synchronization with the Pascal Users Group. As the post office informed you, I recently moved to the address noted. Frankly, I hadn’t received a *Pascal News* in so long that I simply forgot about it. It appears that I won’t miss any issues — the enclosed All-Purpose Coupon is from issue #23.

Interestingly, after a long period (3 years) of not using Pascal, it looks like I will be using it once again. We have a couple of Convergent Technologies workstations in my office. These are very nice 8086-based machines; Burroughs sells them as the B-20s, and NCR sells them as WorkSavers. We will probably be getting a Pascal compiler, and I am looking forward to getting back into Pascaling in the near future.

Sincerely,
Read T. Fleming
144 Irving Avenue #B-3
Providence, RI 02906
November 30, 1982

I was surprised and pleased to receive issue number 24 of Pascal News. Thanks for taking it over. I do have one question, however, which you might be able to help me with. What year is it? My address label includes [82] on it but the previous issue I received was dated September, 1981. I notice that this issue is dated January, 1983. Should I send in another year's subscription money now? What happened to 1982? I never have managed to figure out Pascal News' subscription scheme. Maybe a note in the issues towards the end of a year saying "if your address label says [82] it's time to send in a renewal" would help.

Thanks for your help.

Richard Furuta
Computer Science, FR-35
University of Washington
Seattle, WA 98195

8 February 1983

Dear Sir,

I received your notification of renewal in the mail yesterday. I am slightly concerned that you may not have received the check which I mailed to you in December. I hope that it has only been a slight mix-up, and in fact, my subscription has been renewed for 3 years, as I requested.

I am currently using the Pascal implemented by Microsoft for the IBM Personal Computer. It has some non-standard features which were provided in order to allow programmers to access the full capabilities of the machine. This implementation is quite flexible, and was designed to allow users to produce systems programs, as well as application programs.

The greatest shortcoming to this product, however, is its lack of usable documentation. Even someone like myself, who has been programming in Pascal for 8 years, has difficulty in trying to locate the appropriate material in the 'reference manual'. Once this is overcome, the user is able to use this version for the production of some very powerful software.

I continue to look forward to the delivery of your fine newsletter. I enjoy the articles, and realize how difficult a task you have. Keep up the good work.

Regards,

Robert A. Gibson
1609 Lake Park Dr.
Raleigh, NC 27612

March 3, 1983

Dear Mr. Gaffney:

I'm writing to let you know why I am not renewing my subscription to Pascal News. The main reason is that the price is now too high for the utility of the product (at least to me). I appreciate your efforts to keep PUG and Pascal News going, but I'm afraid they may have outlived their usefulness. Pascal is not really in need of promotion as it was when PUG was formed. The Journal of Pascal & Ada may be an appropriate successor.

As a long-time subscriber and occasional contributor, I wish you luck in your efforts.

Richard Leklanc
Assistant Professor
Georgia Institute of Technology
Atlanta, GA 30332

January 7, 1983

Hang in there, Charlie!

Andy Mickel
106 SE Arthur Avenue
Minneapolis, MN 55414

December 9, 1982

Dear Sirs:

Could you provide us with information on membership in your organization, both personal and institutional, as well as the subscription cost of your journal. We are also interested in a rigorous comparison of the various PASCAL versions implemented by mini and microcomputer vendors. Do you know of any such comparative research? We are making plans to offer Advanced Placement Computer Science in the fall term of 1983, and wish to select an effective computer.

Very truly yours,

Charles McCambridge
Director
Instructional Materials Services
Niskayuna High School
1626 Balltown Rd.
Schenectady, NY 12309

December 25, 1982

Merry Xmas! Good luck, Charlie! Is your "acquisition" of PUG a sign that PUG and USUS will someday merge? I'm not sure I'd like that, but let's see.

Jim Merritt
P.O. Box 1087
Morro Bay, CA 93442

December 24, 1982

Please send me information on joining the Pascal Open Forum

Barbara Huseby, Training Dept.
Electro Scientific Industries
13900 N.W. Science Park Drive
Portland, OR 97229
User’s Group, I am a software project engineer at General Electric in Syracuse. I am currently in the process of selecting a high level language for internal programming of a 1024 × 1280 resolution raster display. Pascal is the leading candidate, therefore, I am very interested in the latest information regarding the language which I feel a user’s group could provide.

My interest does transcend my work however as I do own a Commodore SuperPET which includes the University of Waterloo software package consisting of Pascal, APL, Fortran, Basic and a 6809 Assembler.

Sincerely,
Douglas W. MacDonald
4303 Luna Course
Liverpool, NY 13088

2/5/83
To Whom It May Concern:

I just received your notice to inform me that my membership is about to expire and that I should renew now.

I would like to tell you that I would consider renewing if I could be assured of getting my money’s worth — this time!

When I first joined in 1981, I didn’t hear from Pascal News for almost a year. Then a few months ago, I received a second issue, but that’s been it.

Now I am a convicted Pascaler. I understand the difficulties of operating a non centralized club, but $20 should buy some kind of organization for things I feel.

Can you assure me of a better value this time around?

Cordially,
David Abate
Micro People
116 S. Bowdion St.
Lawrence, MA 01843

P.S. Question: Do you intend anything on UCSD-Pascal? This is my greatest interest.

7 January 1983
Hi,

This is a note in a bottle to: 1) find out if you’re still out there, and 2) what’s happening with Pascal. It doesn’t seem to be taking the bite (or is that byte) out of Basic I thought it would.

We will start covering Pascal as soon as we have finished Basic programming — about five weeks from now. The extension program from Hocking Technical College in Nelsonville has provided seven Apple II and Apple III computers and two printers. By the end of the year, they will have installed a winchester disc and either a modem or a microwave link to their main campus computer. We’ll need it by then to cover the Cobol and Fortran IV programs we’ll be writing.

Most of my practical computer experience is in assembler language. I used it at Cincinnati Milacron’s Process Controls Division (Mater’s of the controls for the T3 Industrial Robot).

I am interested in any literature you have to send me. In particular, I would like the titles of the books you consider best for teaching Pascal — either on the Apple II or on computers in general. Apple, Inc., sent me the Pascal Reference Manual (just a bit or a nibble over my head). I’ve also read copies of the DOS 3.2 Reference Manual and their Basic Programming Manual. I covered all these before classes started and wound up tutoring two other student/inmates.

Sincerely,
Brian Appleman 166-767
15802 St. Rt. 104
P.O. Box 5500
Chillicothe, OH 45601

P.S. If you need more on my background, just ask.

83-02-24
Dear Charlie:

I am a member of PUG (AUS) which has just folded, and I would like to re-enroll through PUG (US).

I don’t share Arthur Sales view that PUG and PN have no purpose now that there is an ISO standard. The world still needs cheap, good software and PN (in a modest way) supplies some of it. Also, some organization is needed to defend and develop good programming language and style.

PUG (AUS) says I have a credit of 12 (old) issues and that the funds have been sent to you. Please will you accept my re-enrollment and advise me how many (new) issues I am now entitled to?

Finally, I, and I’m sure, many others appreciate your offer to keep PUG/PN going.

Thanks again.

Yours sincerely,
Peter Edwards
40 Davison St.
Mitcham, Vic.
Australia 3132

December 3, 1982
Best wishes in this venture, Charlie. I agree that Pascal News and P.U.G. are worth saving.

John W. Baxter
750 State Street, Apt. #224
San Diego, California 92101

February, 1983
You people have ripped me off for the last time!

By your own back order form (attached) you show that my renewal in 1981 paid for 3 issues mailed in 1982. But then, WHAT OF MY RENEWAL PAID IN 1982? ONLY ONE ISSUE #24 COUNTS??? AND THAT HAD TWO PREVIOUSLY PUBLISHED PRO-
GRAMS!! (That is, programs I had ALREADY received.) If you ran a decent organization, you'd make my 1982 renewal count for 1983 also.

David S. Bakin
Softech Inc.
360 Totten Pond Road
Waltham, MA 02154

December 24, 1982
We're indebted to you, Charlie!
Wayne N. Overman
3522 Rockdale Ct.
Baltimore, MD 21207

February 17, 1983
Dear Mr. Gaffney,

I am one of those folks who does not have a currently correct address with Pascal News.

Enclosed is a check for $5 for a copy of issue 19 which was returned to you.

Thank you on behalf of all the members of the user's group for the effort you are putting out. It is very much appreciated.

Tom Bishop
P.O. Box A
Kenmore, WA 98028

March 14, 1983
Dear Sir or Ms.:

We plan to offer Pascal at our school. I would appreciate receiving information on your group and, if possible, a sample copy of Pascal News.

Any suggestions or information you could send would be appreciated. We are particularly concerned that the new Apple 2-E does not support Pascal with one disk drive. We had hoped the UCSD Pascal with one drive would work on the Apple 2-E.

Thanks for your help.

Sincerely,
Harold Baker
Director, Computer Science
Litchfield High School
Litchfield, CT 06759

February 11, 1983
Hi!

Here's my renewal. I really enjoy Pascal News and have been upset about what has happened with it the past 18 months or so. It has been of substantive value to me, particularly in the area of the style of Pascal coding among the community that have submitted articles.

I would like to see more articles on Modula 2, Wirth's follow on to Pascal and Ada in parallel. To me, this would seem a way of keeping PUG alive as well as providing a growth path to these languages for Pascal programmers.

I use Pascal/VS extensively at work and I have found its extensions the best of any other Pascal compiler for S/370 compatible machines. Almost all of its extensions are within the "spirit" of Pascal and uses a very good extension to STRING data. Of particular convenience is its READSTR and WRITESTR functions (they are procedures actually - unfortunately). I force the concept of function upon them by embedding their invocation within a function when required.

I never received issues 20 and 21 of Pascal News during the confusion, although I did mention this at times. I would certainly purchase them separately, but I am not prepared to purchase two sets to get them. Please advise.

Thanks for your work,
Bob Dinah
630 Alvarado St. #207
San Francisco, CA 94114

November 12, 1982
Dear Pascal User's Group:

The only source of information that I have on the Pascal User's Group came from "The BYTE Book of Pascal", according to an article written by Kenneth Bowles. An editor's note of July 1, 1979 listed the annual newsletter as $6.00 per year. I am enclosing $12.00 in case things have increased since that date. If this amount is insufficient, please make it up on back issues.

I am currently using an Apple /// with Apple computer's version of UCSD Pascal. There does not seem to be more than a dozen books written on Pascal, and just a few on UCSD.

I am an ex-electrical engineer, turned to building construction. Previously, I worked for Westinghouse Research Center in Pittsburgh, and used the Burroughs B6500 main frame computer with ALGOL language. The B6500 used a number of formats and types that I miss; the Fixed Format was especially useful since it allowed the user to specify the number of total digits and the number of decimal digits combined. I would like to use this format in UCSD Pascal.

Thanks for taking the time to help me.

Very truly yours,
Larry J. Moorhead
5207 – 32nd Street East
Bradenton, Florida 33508

18 March 1983
Dear Sirs,

For the first time we have received a copy of Pascal News, and it has been read with great interest.

We would like to join your User Group but cannot find either a price or contact address for our region.

Please send us this information as soon as possible,
so that we can become members and start receiving your journal on a regular basis.

We have taken note of your abhorrence of paperwork (and endorse the sentiment) and will send the necessary prepayment once we receive the information.

Yours sincerely,
Bette Kun
Librarian
Control Data
P.O. Box 78105
Sandton, South Africa 2146

20 April 1983
Dear Mr. Shaw:

Enclosed is a check for $10.00 for a one-year subscription to the PASCAL Users’ Group Newsletter. We have just recently acquired PASCAL-2 here at Villanova and our students are using it on LSI-11 systems running RT-11 V4.0 for applications involving real-time control, data acquisition, and computer communications.

Sincerely yours,
Richard J. Perry, Ph.D.
Dept. of Electrical Engineering
Villanova, PA 19085

15th February, 1983
Dear Mr Gaffney,

As a long PUG user the demise of PUG-AUS is a blow. Anyhow, as you can see from the attached letter I would love to continue and thus need your help.

Could you please detail the fees for 1983 for us ‘down under’ for surface mail and air mail and as you can see I’m afraid I’ve not got issue number 21. Can you help?

For interest I use:

UCSD Pascal/p-System
Pascal MT+ under CP/M
and MP/M
Pascal MT+ 86 under CP/M-86
and MP/M-86

Regards,
Dr. William J. Caelli, F.A.C.S.
President
ERACOM Group of Companies
P.O. Box 5488, G.C.M.C.
Qld. 4217, Australia

Dear Mr. Mickel,

I am a student of computing studies in the H.K. Polytechnic. Recently, I got a chance to buy a Chinese version of ‘A Practical Introduction to Pascal’ by Wilson & Addyman from which I was informed that there is a PUG in States.

Briefly understanding the objectives of the PUG, I find myself in great interest in joining the group. Would you be so kind as to provide me with further information as far as the PUG is concerned. I am eagerly looking forward to your reply.

Yours sincerely,
Alan Kwong
12, Boundary St.
Po Hing Bldg.
8/F, Block ‘C’
Kln., H.K.
December 23, 1982
We have been using Oregon Software’s RT-11 Pascal implementations for over three years with excellent results and complete satisfaction; Pascal is used for scientific "number crunching", program development, algorithm testing, etc.

Bob Schor
The Rockefeller University
1230 York Avenue
New York, NY 10021

December 30, 1982
A worthwhile journal.

George Williams
Union College
Schenectady, NY 12308

March 22, 1983
Dear Mr. Gaffney:
I have previously received Pascal News through University of Tasmania. Is it still published? If so, do I have any credit on my subscription dues? I would also be interested in information about USUS.

Yours sincerely,
M.J. Palmer
CSIRO
Private Bag
P.O. Wembley, W.A. 6014

February 3, 1983
Good job, Charlie! and good luck to the renewed Pascal News!

Norman W. Molhant
320 Principale
Tres-Saint-Redempteur, P.Q.
Canada JOP 1P0

May 1, 1983
A professor in Ithaca, NY told me there exists a public domain UCSD Pascal available for micro's.
I have a 60K Z-80 which uses memory map video, and a 63K 8085/8088 (both machines S-100 bus) which uses a TVI 950. I also have a H-29 terminal (like Z-19 but with a detached keyboard).
Is there really any way of getting this UCSD Pascal running on one of my systems? (I have UCSD on the Sage also Modula-2. Good stuff.)

Thanks,
J. E. Pournelle, Ph.D.
12051 Laurel Terrace
Studio City, CA 91604
Program APLscanner

By Vincent Dichristofano, Alan Kaniss, Thomas Robinson, and John Santini
NADC, Philadelphia, PA

1 program APLscanner(input | * TERMINAL *, output , APLfile | $);
2
3 | * Purpose: *
4 | This program is an implementation of APL in Pascal.
5
6 | * Authors: *
7 | Vincent Dichristofano
8 | Alan Kaniss
9 | Thomas Robinson
10 | John Santini
11
12 | authors' affiliation - NADC
13 | \[ Phil. PA, USA \]
14
15 | project leader: Dr. Joseph Mezzanotte
16
17 | This program was written as part of an independent study
18 | course at Villanova University.
19
20 | * Submitted and accepted for Pascal News. TEC 1976. *
21
22 | label
23 | $100$;
24
25 const
26 prefix = 60;
27 | prefix = 62 | ( prefix for CCI ASCII 12-bit codes )
28 MaxVarNameLength = 10;
29 MaxInputLine = 113;
30 MaxInputArraySize = 114;
31 NumberOfMessages = 100;
32 MessageLength = 80;
33
34 type
35 PackedString = packed array [1 .. MaxVarNameLength] of 0 .. 8191;
36 TokenName =
37 (foreName, foreAge, GlobVar, MonAdOper, ReductOper, DyadOper,
38 SpecOper, constant, StatEnd);
39 values = record
40 | RealValue: real;
41 | NextValue: values
42 | end
43 VarTab = record
44 | VarName: PackedString [v];
45 | FuncPtr: "vfunc [v - vtab ;]
46 | ValTabPtr: "ValTab [v - vtab ;]
47 | deferredVarTabPtr: "fParTab;
48 | NextVarTab: "VarTab;
49 | end
50 ValTab = record
51 | record
52 | NextToken: "TokenTable;
53 | case name: TokenName of
54 | (FormRes, foreName, GlobVar: "stab | var | (VarTabPtr. "VarTab;
55 | MonAdOper: (MonIndex | integer);
56 | ReductOper: RedIndex | integer);
57 | DyadOper: (DyopIndex | integer);
58 | SpecOper: (CharIndex | integer);
59 | constants: (Vtab | TabVal);
60 | ConstTab: "ConstTab (EndIndex | integer);
61 | end;
62 | vfunc = record
63 | NextToken: "TokenTable;
64 | NextFuncPtr: "vfunc;
65 | StatLabel: PackedString
66 | end
67 | OperatorType = (niladic, monadic, dyadic);
68 | FuncTab = record
69 | function: PackedString [f1];
70 | arity: OperatorType | s2 | integer
71 | result: Boolean | s3 | true = explicit | f3 | f4 | f5; 12
72 | LeftArg: PackedString | [f6 ;]
73 | RightArg: PackedString | [f7 ;]
74 | FirstStatement: "vfunc;
75 | NextFuncPtr: "vfunc;
76 | NumOfStatements: integer
77 | end
78 fParTab = record
79 | FParVal: "ValTab [sd1 and sD2 ;]
80 | LastPar: "fParTab [link to last sD2 ;]
81 | end;
82 dienInfo = record
83 | NextDien: "dienInfo;
84 | dienLength: integer
85 | end
86 OpRecord = record
87 | OpIndex: integer;
88 | OpSymbol: integer;
89 | end
90 Operand = record
91 | OperPtr: "ValTab [sD2 ;]
92 | NextOper: "Operand (link to last sD2 ;)
93 | end
94 SubTab = record
95 | CalledSubr: "vfunc [sD1 ;]
96 | TokenCallingSubr: "TokenTable [sd1 ;]
97 | StamenCallingSubr: "vfunc [sD1 ;]
98 | LastSubs: SubTab (link to last sD1 ;)
99 | end
100 OpTable = array [1 .. 163] of OpRecord;
101 TypeValTable = "ValTab;
102 TypeFuncTab = "FuncTab;
103 TypeValuePtr = "values;
104 APLcharSet =
105 (xsymbol, XSymbol, Symbol, SSymbol, SSymbol, TSymbol, USymbol,
106 XSymbol, SSymbol, TSymbol, USymbol, VSymbol, VSymbol,
107 ZSymbol, ZSymbol, OneSymbol, TwoSymbol, ThreeSymbol, FourSymbol, FiveSymbol, SixSymbol, SevenSymbol,
108 EightSymbol, NineSymbol, ZeroSymbol, colon, RightArrow, LeftArrow,
109 SmallCircle, period, LeftParen, RightParen, LeftBracket,
110 RightBracket, semicolon, quadrant, space, plus, minus, times,
111 divide, asterisk, plus, comma, tilde, equals, NotEqual,
112 LessThan, LessOrEqual, GreaterThan, GreaterOrEqual, AndSymbol,
113 OrSymbol, ceiling, floor, LargeCircle, ForwardSlash, DoubleQuote,
114 negative, QuestionMark, omega, epsilon, UpArrow, DownArrow, alpha,
115 UnderScore, del, delta, SingleQuote, EastCap, WestCap, SouthCap,
116 NorthCap, ibeam, Tbeam, VerticalStroke, BackwardSlash);
117 text = file of char;
118
119 var
120 XCircle, XLeftArrow, XRightArrow, XLittleCircle, XPeriod, XLeftPar,
121 XRightPar, XLeftBracket, XRightBracket, XSemantic, XKubeSym,
122 integer;
123 character: array [APLcharSet] of integer;
124 APLStates: array [1 .. InputArraySize] of integer;
125 digits: array [OneSymbol, ZeroSymbol] of integer;
126 ErrorMessage: packed array [1 .. NumberOfMessages, 1 .. MessageLength] of char;
127 APLlexer: text;
128 MoTab, DoTab, RedTab, CharTab, SpecTab: OpTable;
129 SaveLabel: PackedString;
130 name: PackedString;
131 NewTokenPtr, OldTokenPtr, HoldTokenPtr, SaveTokenPtr: "TokenTable;
132 TestfuncTable, NewfuncTable, OldfuncTable: "FuncTab;
133 NewVarTabPtr, OldVarTabPtr: "VarTab;
134 LeftValPtr, RightValPtr, ValPtr: "values;
135 NewValues, NewVarPtr: "values;
136 Dim: "DimInfo;
137 DimPtr, NewPtr, LeftDimPtr, RightDimPtr: "DimInfo;
138 varValue: "VarTab;
139 OldFunPtr, NewFUnPtr: "func;
140 NewValTabLink, OldValTabLink: "ValTab;
141 position: integer;
142 LineLength: integer;
143 code, CoLnt: integer;
144 FunTabStates: integer;
145 TokenError: FirstFunction: Boolean;
146 LineToolong, HasLabel: Boolean;
147 switch, FunctionMode, TokenSwitch, ItsKIdentifier: Boolean;
148 OperTabPtr: "Operand (sv 1 ;)
procedure InitializeCharacterSet;

begin
  var TestForPrefix: integer;
  fileCharacter: char;
  SymbolIndex: APLcharSet;

  begin
    reset( APLfile); for SymbolIndex := to symbol to BackwardSlash do
      begin
        TestForPrefix := ord( fileCharacter); if ( TestForPrefix = prefix1) or ( TestForPrefix = prefix2) then
          begin
            read( APLfile, fileCharacter);
            SymbolIndex := SymbolIndex * 100 + TestForPrefix + ord( fileCharacter);
          end
        else
          SymbolIndex := ord( fileCharacter);
      end;

    end;

  var MsgRow, MsgCol: integer;

begin
  read( APLfile); for MsgRow := 1 to NumberOfMessages do
    for MsgCol := 1 to MessageLength do
      begin
        ErrorMsg[ MsgRow, MsgCol] := 0;
        read( APLfile, ErrorMsg[ MsgRow, MsgCol]);
      end;

end;

procedure PrintAPLStatement;

begin

  var prefix: num;

begin
  index := 1 to LineLength do
    begin
      if ( APLstatement[ index] <= 100) then
        begin
          prefix := chr( chr( APLstatement[ index] + 100));
          write( prefix);
          num := chr( chr( APLstatement[ index] - 100));
          write( num);
        end
      else
        write( num);

    end;
end;
procedure GetAPLstatement;

var
  InputChar: char;
  TestForPrefix: integer;
  FirstTry: Boolean;
  sign: integer;
  DigitCount: integer;
  LineLength: integer;
  NameOne: strings;
  NameToo: strings;
  TableIndex: integer;
  test: string;

begin
  if FirstTry then getinput(input); // test for * only

  FirstTry := false;
  while (not eoln(input)) and (not LineTooLong) do
    if LineLength < MaxInputLine then
      begin
        LineLength := LineLength + 1; readInputChar;
        if (sign = 1) then begin
          sign := -1; position := position + 1;
        end
        else
          begin
            APLstatement(LineLength) := ord(InputChar);
            if LineTooLong then error(1);
          end;
      end
    end
    else
      begin
        if LineLength < MaxInputLine then
          begin
            APLstatement(LineLength) := ord(InputChar);
            if LineTooLong then error(1);
          end;
        if (sign = 1) then begin
          sign := -1; position := position + 1;
        end
        else
          begin
            APLstatement(LineLength) := ord(InputChar);
            if LineTooLong then error(1);
          end;
      end
    end
end;
function MonadicReference: Boolean;

var
  SubPosition: TableIndex: integer;

begin { see if operator is monadic within context of input line }
  if NewTokenPtr'^.NextToken'.noun = StatEnd
  then MonadicReference := true

else begin
  SubPosition := position + 1;

  while (SubPosition > 0) and (APLStatement[SubPosition] = character(=space)) do
    SubPosition := SubPosition - 1; { get last non-blank }

  if SubPosition < 0 then
    TableLookup(APLStatement[SubPosition], 6, SpecTab, TableIndex)
  else if (TableIndex < 0) or (SubPosition = 0)
    then MonadicReference := true

  else begin
    if (NewTokenPtr'^.NextToken'.noun = FormRes)
      and (NewTokenPtr'^.NextToken'.noun = FormArg)
      and (NewTokenPtr'^.NextToken'.noun = constant)
      and (APLStatement[SubPosition] = character(=period))
      and (APLStatement[SubPosition] = character(=character))
      and (RightPare=) and (APLStatement[SubPosition] = character(=character))
      and (RightBracket)
      then MonadicReference := true

    else begin
      if (NewTokenPtr'^.NextToken'.noun = FormRes)
        and (NewTokenPtr'^.NextToken'.noun = FormArg)
        and (NewTokenPtr'^.NextToken'.noun = constant)
        and (APLStatement[SubPosition] = character(=period))
        and (APLStatement[SubPosition] = character(=character))
        and (RightPare=) and (APLStatement[SubPosition] = character(=character))
        and (RightBracket)
      then MonadicReference := true

      end

    end
  end

end

begin { check other tables }

  if TableIndex = -1
  then
     begin 
    ChkIndex := NewValTabPtr'^.FirstValue'
    if not ItsANumber then CheckOtherTables
    if not ItsANumber then CheckOtherTables
    else
     begin
       NumberCount := 0; MakeNumber(RealNumber, ItsANumber)
       if not ItsANumber then CheckOtherTables
     end
  else
   begin
      numberCount := 0; NewNumber := RealNumber
      if not ItsANumber then CheckOtherTables
      else
        begin
          NewNumber := NewNumber + NewValues
          if not ItsANumber then CheckOtherTables
        end
  end

end

else if MonadicReference
  then begin
      MonadicReference := true
    end

else begin
  if not MonadicReference
    then begin
    if operator is valid reduction operator
      then NewTokenPtr'^.RedIndex' := TableIndex
    end
  end

if position = ChkIndex + 1
  then begin
    if checkothertables
    then begin
      TableLookup(APLStatement[position], 9, MOpTab, TableIndex)
      if TableIndex = 0 then DyadicOpCheck
    end
  end

else begin
  if not MonadicReference
    then begin
    if operator is monadic
      then NewTokenPtr'^.NonIndex' := TableIndex
    end
  end

if position := position + 1
  then begin
    checkothertables
  end

end

begin

  if not MonadicReference
    then begin
    if operator is valid reduction operator
      then NewTokenPtr'^.RedIndex' := TableIndex
    end

end

begin

  if (NewTokenPtr'^.NextToken'.noun = 'MonadOp')
    and (NewTokenPtr'^.NonIndex' := TableIndex).

end

begin

  if position = ChkIndex + 1
  then begin
    if checkothertables
    then begin
      TableLookup(APLStatement[position], 9, MOpTab, TableIndex)
      if TableIndex = 0 then DyadicOpCheck
    end
  end

else begin
  if not MonadicReference
    then begin
    if operator is monadic
      then NewTokenPtr'^.NonIndex' := TableIndex
    end
  end

if position := position + 1
  then begin
    checkothertables
  end

end

begin

  if not MonadicReference
    then begin
    if operator is valid reduction operator
      then NewTokenPtr'^.RedIndex' := TableIndex
    end

end

begin

  if (NewTokenPtr'^.NextToken'.noun = 'MonadOp')
    and (NewTokenPtr'^.NonIndex' := TableIndex).

end

begin

  if position = ChkIndex + 1
  then begin
    if checkothertables
    then begin
      TableLookup(APLStatement[position], 9, MOpTab, TableIndex)
      if TableIndex = 0 then DyadicOpCheck
    end
  end

else begin
  if not MonadicReference
    then begin
    if operator is monadic
      then NewTokenPtr'^.NonIndex' := TableIndex
    end
  end

if position := position + 1
  then begin
    checkothertables
  end

end

begin

  if not MonadicReference
    then begin
    if operator is valid reduction operator
      then NewTokenPtr'^.RedIndex' := TableIndex
    end

end

begin

  if (NewTokenPtr'^.NextToken'.noun = 'MonadOp')
    and (NewTokenPtr'^.NonIndex' := TableIndex).

end

begin

  if position = ChkIndex + 1
  then begin
    if checkothertables
    then begin
      TableLookup(APLStatement[position], 9, MOpTab, TableIndex)
      if TableIndex = 0 then DyadicOpCheck
    end
  end

else begin
  if not MonadicReference
    then begin
    if operator is monadic
      then NewTokenPtr'^.NonIndex' := TableIndex
    end
  end

if position := position + 1
  then begin
    checkothertables
  end

end

begin

  if not MonadicReference
    then begin
    if operator is valid reduction operator
      then NewTokenPtr'^.RedIndex' := TableIndex
    end

end

begin

  if (NewTokenPtr'^.NextToken'.noun = 'MonadOp')
    and (NewTokenPtr'^.NonIndex' := TableIndex).

end

begin

  if position = ChkIndex + 1
  then begin
    if checkothertables
    then begin
      TableLookup(APLStatement[position], 9, MOpTab, TableIndex)
      if TableIndex = 0 then DyadicOpCheck
    end
  end

else begin
  if not MonadicReference
    then begin
    if operator is monadic
      then NewTokenPtr'^.NonIndex' := TableIndex
    end
  end

if position := position + 1
  then begin
    checkothertables
  end

end

begin

  if not MonadicReference
    then begin
    if operator is valid reduction operator
      then NewTokenPtr'^.RedIndex' := TableIndex
    end

end

begin

  if (NewTokenPtr'^.NextToken'.noun = 'MonadOp')
    and (NewTokenPtr'^.NonIndex' := TableIndex).

end

begin

  if position = ChkIndex + 1
  then begin
    if checkothertables
    then begin
      TableLookup(APLStatement[position], 9, MOpTab, TableIndex)
      if TableIndex = 0 then DyadicOpCheck
    end
  end

else begin
  if not MonadicReference
    then begin
    if operator is monadic
      then NewTokenPtr'^.NonIndex' := TableIndex
    end
  end

if position := position + 1
  then begin
    checkothertables
  end

end

begin

  if not MonadicReference
    then begin
    if operator is valid reduction operator
      then NewTokenPtr'^.RedIndex' := TableIndex
    end

end

begin

  if (NewTokenPtr'^.NextToken'.noun = 'MonadOp')
    and (NewTokenPtr'^.NonIndex' := TableIndex).

end

begin

  if position = ChkIndex + 1
  then begin
    if checkothertables
    then begin
      TableLookup(APLStatement[position], 9, MOpTab, TableIndex)
      if TableIndex = 0 then DyadicOpCheck
    end
  end

else begin
  if not MonadicReference
    then begin
    if operator is monadic
      then NewTokenPtr'^.NonIndex' := TableIndex
    end
  end

if position := position + 1
  then begin
    checkothertables
  end

end
805 if NewTokenPtr <> nil then
806 then (NewTokenPtr', nnn = FormRes) or (NewTokenPtr'.noum = FormArg
807 then (NewVarTabPtr'.FuncTabPtr = NewFuncTabPtr
808 else NewVarTabPtr'.FuncTabPtr = nil
809 end | adddomevariable) |;
810
811 function FunctionAlreadyDefined(var NewFuncName: PackedString; var
812       FuncIndex: PrFuncTab; Boolean)
813 var
814   DummyPt: FuncTab;
815   name1, name2, name3: PackedString;
816   ItsAnIdentifier, FuncHeadError: Boolean;
817 end
818
819 procedure MakeTokenLink;
820 begin
821 NewTokenPtr := NewTokenPtr'.NextToken := OldTokenPtr;
822 OldTokenPtr := NewTokenPtr;
823 end
824
825 procedure ProcessfunctionHeader;
826 var
827   DummyPtr: FuncTab;
828   name1, name2, name3: PackedString;
829   ItsIdentifier, FuncHeadError: Boolean;
830   ArityIndex: integer;
831
832 function FunctionError: false; FunctionMode := true;
833 function Statements := 1;
834 if FirstFunction then begin
835   if Statements := 0; else begin
836     if not ItsAnIdentifier then begin
837       identifier(name1, ItsAnIdentifier);
838       if not ItsAnIdentifier then
839         then (begin
840         if (error(T)) then begin
841           if not ItsAnIdentifier then
842             then (begin
843               if FirstFunction then begin
844                 if Statements := 0; else begin
845                   if not ItsAnIdentifier then
846                     then (begin
847                       Error(T) | unrecognizable function's argument name |;
848                       FunctionMode := false | exit function mode |;
849                       FuncHeadError := true
850                     end)
851                     else (begin
852                       NewVarTabPtr := SkipSpaces;
853                       if APLStatement(position) = character(LeftArrow) then
854                       then (begin
855                         NewFuncTabPtr'.result := true | explicit result |;
856                         NewFuncTabPtr'.ResultName := name1;
857                         position := position + 1;
858                         identifier(name1, ItsIdentifier);
859                         if not ItsIdentifier then
860                           then (begin
861                             Error(T) | invalid function's argument name |;
862                             FunctionMode := true
863                           end)
864                       end)
865                       else (begin
866                         if (position <= LineLength) and (not FuncHeadError) then
867                         then (begin
868                           identifier(name2, ItsIdentifier);
869                         end)
860                       end)
869                       else (begin
870                         identifier(name3, ItsIdentifier);
871                         if not ItsIdentifier then
872                         then (begin
873                           Error(T) | invalid function right argument name |;
874                         end)
875                         if (position <= LineLength) and (not FuncHeadError) then
876                         end)
877                   end)
878                 end)
879
880 if (position <= LineLength) and (not FuncHeadError) then
881 begin
882   Error(T) | extraneous characters to right of function header |;
883   FuncHeadError := true
884 end
885
886 if ArityIndex = 1 then
887 begin
888   NewFuncTabPtr'.arity := niladic;
889   NewFuncTabPtr'.FuncName := name1;
890 end
891
892 if ArityIndex = 2 then
893 begin
894   NewFuncTabPtr'.arity := monadic;
895   NewFuncTabPtr'.FuncName := name1;
896   NewFuncTabPtr'.RightArg := name2;
897   AddNameToVarTable(name2);
898   NewFuncTabPtr'.FuncTabPtr := NewFuncTabPtr;
899 end
900
901 if ArityIndex = 3 then
902 begin
903   NewFuncTabPtr'.arity := dyadic;
904   NewFuncTabPtr'.LeftArg := name1;
905   NewFuncTabPtr'.FuncName := name1;
906   NewFuncTabPtr'.RightArg := name3;
907   AddNameToVarTable(name3);
908   NewFuncTabPtr'.FuncTabPtr := NewFuncTabPtr;
909 end
910
911 if (not FuncHeadError) then begin
912   dispose(NewFuncTabPtr);
913   FunctionMode := true | exit function mode |;
914   NewFuncTabPtr := OldFuncTabPtr;
915 end
916
917 if (error(T)) then begin
918     if not FuncHeadError then begin
919       dispose(NewFuncTabPtr);
920       FunctionMode := true | exit function mode |;
921     end)
922     if (error(T)) then begin
923       dispose(NewFuncTabPtr);
924       FunctionMode := true | exit function mode |;
925     end)
926     if (error(T)) then begin
927       dispose(NewFuncTabPtr);
928       FunctionMode := true | exit function mode |;
929     end)
930     if (error(T)) then begin
931       dispose(NewFuncTabPtr);
932       FunctionMode := true | exit function mode |;
933     end)
934     if (error(T)) then begin
935       dispose(NewFuncTabPtr);
936       FunctionMode := true | exit function mode |;
937     end)
938     if (error(T)) then begin
939       dispose(NewFuncTabPtr);
940       FunctionMode := true | exit function mode |;
941     end)
942     if (error(T)) then begin
943       dispose(NewFuncTabPtr);
944       FunctionMode := true | exit function mode |;
945     end)
946     if (error(T)) then begin
947       dispose(NewFuncTabPtr);
948       FunctionMode := true | exit function mode |;
949     end)
950     if (error(T)) then begin
951       dispose(NewFuncTabPtr);
952       FunctionMode := true | exit function mode |;
953     end)
954     if (error(T)) then begin
955       dispose(NewFuncTabPtr);
956       FunctionMode := true | exit function mode |;
957     end)
958     if (error(T)) then begin
959       dispose(NewFuncTabPtr);
960       FunctionMode := true | exit function mode |;
961     end)
962     if (error(T)) then begin
963       dispose(NewFuncTabPtr);
964     end)
965     if (error(T)) then begin
966     end)
967     if (error(T)) then begin
968     end)
969     if (error(T)) then begin
970     end)
971     if (error(T)) then begin
972     end)
973     if (error(T)) then begin
974     end)
975     if (error(T)) then begin
976     end)
977     if (error(T)) then begin
978     end)
979     if (error(T)) then begin
980     end)
function SpecSymbol(sym: integer): Boolean;

procedure error(index: integer);

begin { return to calling function }

if SubrTabPtr.NavigateSubr;

end (calledSubr);

begin (callSubr)

if not NameInVarTable(SubTabPtr, CalledSubr, RightArg, PtrToVarTab, SubTabPtr, CalledSubr) then error(32);

if PtrToVarTab.FuncTabPtr <> SubTabPtr, CalledSubr

begin (if dyadic)

if not NameInVarTable(SubTabPtr, CalledSubr, LeftArg, PtrToVarTab, SubTabPtr, CalledSubr) then error(33);

if error(33) then same as error(32) |

end

callSubr;

end (callSubr)

function funcCall: Boolean;

begin (funcCall)

false;

if TokenTabPtr.Name = GlobalVar

begin (if defined)

true;

end (func call);

begin (nullcall)

begin (return to calling function)

VFuncPtr := SubTabPtr, StatedCallingSubr;

TokenTag := SubTabPtr, TokenCallingSubr, NextToken;

if SubTabPtr, CalledSubr, arity <> niladic

then

begin (dyadic only)

AuxLParmPtr := LParmPtr;

LParmPtr := LParmPtr, LastParm;

dispose(AuxLParmPtr);

end

end (returnToCallingSubr)

begin {symbol not found}

false;

else

true;

end (error)


}
function variable: Boolean;

begin (inputval)

var

GlobOrDummy: Boolean; gond;
PassedAdj: VarTab[k];
ParmPtr: VarTab[pt];
ValVar: Boolean;
ValIndex: Boolean;

procedure Inputval;

var

AuxPtrToTab: ValTab;
AuxValuesPtr: values;
AuxVarValuesPtr: values;
RealVal: real;
bool: Boolean;
integer;
AuxDimenF0Ptr: DimenInfo;

begin (inputval)

for cnt := 0 to MessageLength do write(ErrorMsgs[65, cnt2]);
readln; GetErrorMsgs;
repeat
MaxNumberOfRealV, bool); SkipSpaces;
if not bool then
begin
for ColInt := 1 to MessageLength do
write(ErrorMsgs[65, ColInt]);
writing: position := 1; cnt := 0;
AuxValuesPtr := OperTabPtr'. OperPtr'. FirstValue;
for cnt := 1 to MessageLength do
write(ErrorMsgs[65, cnt]);
writing: readln; GetErrorMsgs;
end
else
begin
var

cnt := cnt + 1; AuxValuesPtr := AuxValuesPtr;
new(AuxVarValuesPtr); AuxValuesPtr'. RealVal := RealV;
AuxValuesPtr'. NextValue := AuxValuesPtr;
end;

until position > LineLength;
dispose(AuxValuesPtr); AuxValuesPtr'. NextValue := nil;
PtrTab'. IntermedResult := false; PtrTab'. dimensions := 1;
PtrTab'. ForwardOrder := true;
unless newAuxDimenF0Ptr;
AuxDimenF0Ptr'. dimensions := cnt;
AuxDimenF0Ptr'. NextValue := nil;
end (inputval);

procedure GetArrayPosition(var ValuesPtr: TypeValuesPtr;

var

indice: real;
kont: integer;
sl: integer;
AuxDimenF0Ptr: DimenInfo;

begin (getarrayposition)

if npv <> ParmPtr'. dimensions then error(35);
if "wrong number of subscripts".
sl := 0; AuxOperand := OperTabPtr;
AuxDimenF0Ptr := ParmPtr'. FirstDimen;
for kcnt := 1 to npv do

begin

if AuxOperand'. OperPtr'. dimensions <> 0 then error(35);
then error(35) ("out of range index");
sl := sl + AuxDimenF0Ptr'. dimensions + trunc(indice) <= 0.
AuxOperand := OperTabPtr;
AuxOperand := AuxOperand'. FirstDimen;
AuxOperand := AuxOperand'. FirstDimen;
end;

ValuesPtr := ParmPtr'. FirstValue;
while sl <> 0 do (determine which value in )

pt[sval(sval(sval(sl)))] sval(sval(sval(sl)));
sval(sval(sl));

begin (getarrayposition)

procedure LinkResults;

var

PtrToValues: values;

begin (linkresults)

if npv = 0 then

begin

if globOrDummy then

begin

if rang then ParmPtr'. PtrVal := OperTabPtr'. OperPtr;
else LParmPtr'. PtrVal := OperTabPtr'. OperPtr;
else PassedAdj'. ValTabPtr := OperTabPtr'. OperPtr;
end;

end;

begin

for cnt := 1 to MessageLength do write(ErrorMsgs[65, cnt2]);
readln; GetErrorMsgs;
repeat
MaxNumberOfRealV, bool); SkipSpaces;
if not bool then
begin
for ColInt := 1 to MessageLength do
write(ErrorMsgs[65, ColInt]);
writing: position := 1; cnt := 0;
AuxValuesPtr := OperTabPtr'. OperPtr'. FirstValue;
for cnt := 1 to MessageLength do
write(ErrorMsgs[65, cnt]);
writing: readln; GetErrorMsgs;
end
else
begin
var

cnt := cnt + 1; AuxValuesPtr := AuxValuesPtr;
new(AuxVarValuesPtr); AuxValuesPtr'. RealVal := RealV;
AuxValuesPtr'. NextValue := AuxValuesPtr;
end;

until position > LineLength;
dispose(AuxValuesPtr); AuxValuesPtr'. NextValue := nil;
PtrTab'. IntermedResult := false; PtrTab'. dimensions := 1;
PtrTab'. ForwardOrder := true;
unless newAuxDimenF0Ptr;
AuxDimenF0Ptr'. dimensions := cnt;
AuxDimenF0Ptr'. NextValue := nil;
end (inputval);

procedure GetArrayPosition(var ValuesPtr: TypeValuesPtr;
function SimpleVariable: Boolean;

begin

ValidSv := false;

if not SimpleVariable

then begin

if (not Val idIndex) or (not SpecSymboL(XLeftBracket))

then rarg := true;

end;

else begin

if (not Val idIndex) or (not SpecSymboL(XLeftPar))

then rarg := true;

end;

end;

end

function SimpleVariable: Boolean;

var

ValidSv: Boolean;

begin

ValidSv := false;

globVarDummy := false;

if assign

then begin

begin

if (TokenTabPtr ".noun = FormRes) or (TokenTabPtr ".noun = FormArg)

then begin

end;

end;

begin

end;

begin

end;

begin

end;

end;

end;

begin

valid := false; expression(ValidE1);

if ValidE1

then

begin

begin

end;

end;

end;

begin

ValidVar := false; npv:=

if not assign

then begin

end;

end;

begin

begin

end;

begin

end;

begin

end;

begin

end;

begin

end;

begin

end;

begin

end;

begin

end;

begin

end;

begin

end;

begin

end;

begin

end;

begin

end;

begin

end;

begin

end;

begin

end;

begin

end;

begin

end;

begin

end;
not FunctCall then valid := false

procedure expression | recursive |;

var
 DoneExp, Value, ValidAss: Boolean;
code: integer;

procedure assignment | var: valid: Boolean;

begin ( assignment )
valid := false;
if Symbol(XLeftArrow) then begin
assign := true; assign1 := true;
end;
else error(8) Result of an assn not a valid variable |;
valid := true; assign := false;
end;

function mop: Boolean;

begin ( mop )
valid := false;
if (TokenTabPtr').nnum = MonadOper) or (TokenTabPtr').nnum = SubtractOper) then begin
TokenTabPtr'.nnum = MonadOper) or (TokenTabPtr'.nnum = SubtractOper) then begin
TokenTabPtr'.nnum = MonadOper) or (TokenTabPtr'.nnum = SubtractOper) then begin
TokenTabPtr' := TokenTabPtr'.NextToken; ValidID := true;
map := ValidID;
end ( mop )

function dop: Boolean;

validate := true;
if (TokenTabPtr').nnum = DyadOper then begin
begin ( dop )
end;

begin ( dop )
end;

end ( dop )

end;
end;
end;
procedure monadic(arg: TypeValTabPtr; token: TokenPtr);

begin
  if switch
    NewVal := arg;
    while ValPtr <> nil do
      if switch
        NewVal := ValPtr;
        while ValPtr <> nil do
          if switch
            NewVal := ValPtr;
            while ValPtr <> nil do
              if switch
                NewVal := ValPtr;
                while ValPtr <> nil do
                  if switch
                    NewVal := ValPtr;
                    while ValPtr <> nil do
                      if switch
                        NewVal := ValPtr;
                        while ValPtr <> nil do
                          if switch
                            NewVal := ValPtr;
                            while ValPtr <> nil do
                              if switch
                                NewVal := ValPtr;
                                while ValPtr <> nil do
                                  if switch
                                    NewVal := ValPtr;
                                    while ValPtr <> nil do
                                      if switch
                                        NewVal := ValPtr;
                                        while ValPtr <> nil do
                                          if switch
                                            NewVal := ValPtr;
                                            while ValPtr <> nil do
                                              if switch
                                                NewVal := ValPtr;
                                                while ValPtr <> nil do
                                                  if switch
                                                    NewVal := ValPtr;
                                                    while ValPtr <> nil do
                                                      if switch
                                                        NewVal := ValPtr;
                                                        while ValPtr <> nil do
                                                          if switch
                                                            NewVal := ValPtr;
                                                            while ValPtr <> nil do
                                                              if switch
                                                                NewVal := ValPtr;
                                                                while ValPtr <> nil do
                                                                  if switch
                                                                    NewVal := ValPtr;
                                                                    while ValPtr <> nil do
                                                                      if switch
                                                                        NewVal := ValPtr;
                                                                        while ValPtr <> nil do
                                                                          if switch
                                                                            NewVal := ValPtr;
                                                                            while ValPtr <> nil do
                                                                              if switch
                                                                                NewVal := ValPtr;
                                                                                while ValPtr <> nil do
                                                                                  if switch
                                                                                    NewVal := ValPtr;
                                                                                    while ValPtr <> nil do
                                                                                      if switch
                                                                                        NewVal := ValPtr;
                                                                                        while ValPtr <> nil do
                                                                                          if switch
                                                                                            NewVal := ValPtr;
                                                                                            while ValPtr <> nil do
                                                                                              if switch
                                                                                                NewVal := ValPtr;
                                                                                                while ValPtr <> nil do
                                                                                                    if switch
                                                                                                     NewVal := ValPtr;
                                                                                                     while ValPtr <> nil do
                                                                                                       if switch
                                                                                                        NewVal := ValPtr;
                                                                                                        while ValPtr <> nil do
                                                                                                          if switch
                                                                                                            NewVal := ValPtr;
                                                                                                            while ValPtr <> nil do
                                                                                                              if switch
                                                                                                                NewVal := ValPtr;
                                                                                                                while ValPtr <> nil do
                                                                                                                  if switch
                                                                                                                    NewVal := ValPtr;
                                                                                                                    while ValPtr <> nil do
                                                                                                                      if switch
                                                                                                                        NewVal := ValPtr;
                                                                                                                        while ValPtr <> nil do
                                                                                                                          if switch
                                                                                                                            NewVal := ValPtr;
                                                                                                                            while ValPtr <> nil do
                                                                                                                              if switch
                                                                                                                                NewVal := ValPtr;
                                                                                                                                while ValPtr <> nil do
                                                                                                                                  if switch
                                                                                                                                    NewVal := ValPtr;
                                                                                                                                    while ValPtr <> nil do
                                                                                                                                      if switch
                                                                                                                                        NewVal := ValPtr;
                                                                                                                                        while ValPtr <> nil do
                                                                                                                                          if switch
                                                                                                                                            NewVal := ValPtr;
                                                                                                                                            while ValPtr <> nil do
                                                                                                                                             if switch
                                                                                                                                             NewVal := ValPtr;
                                                                                                                                             while ValPtr <> nil do
                                                                                                                                             if switch
                                                                                                                                             NewVal := ValPtr;
                                                                                                                                             while ValPtr <> nil do
                                                                                                                                             if switch
                                                                                                                                             NewVal := ValPtr;
                                                                                                                                             while ValPtr <> nil do
                                                                                                                                             if switch
                                                                                                                                             NewVal := ValPtr;
                                                                                                                                             while ValPtr <> nil do
                                                                                                                                             if switch
                                                                                                                                             NewVal := ValPtr;
                                                                                                                                             while ValPtr <> nil do
                                                                                                                                             if switch
                                                                                                                                             NewVal := ValPtr;
                                                                                                                                             while ValPtr <> nil do
                                                                                                                                             if switch
                                                                                                                                             NewVal := ValPtr;
                                                                                                                                             while ValPtr <> nil do
                                                                                                                                             if switch
                                                                                                                                             NewVal := ValPtr;
                                                                                                                                             while ValPtr <> nil do
                                                                                                                                             if switch
                                                                                                                                             NewVal := ValPtr;
                                                                                                                                             while ValPtr <> nil do
                                                                                                                                             if switch
                                                                                                                                             NewVal := ValPtr;
                                                                                                                                             while ValPtr <> nil do
                                                                                                                                             if switch
                                                                                                                                             NewVal := ValPtr;
                                                                                                                                             while ValPtr <> nil do
                                                                                                                                             if switch
                                                                                                                                             NewVal := ValPtr;
                                                                                                                                             while ValPtr <> nil do
                                                                                                                                             if switch
                                                                                                                                             NewVal := ValPtr;
                                                                                                                                             while ValPtr <> nil do
                                                                                                                                             if switch
                                                                                                                                             NewVal := ValPtr;
                                                                                                                                             while ValPtr <> nil do
                                                                                                                                             if switch
                                                                                                                                             NewVal := ValPtr;
                                                                                                                                             while ValPtr <> nil do
                                                                                                                                             if switch
                                                                                                                                             NewVal := ValPtr;
                                                                                                                                             while ValPtr <> nil do
                                                                                                                                             if switch
                                                                                                                                             NewVal := ValPtr;
                                                                                                                                             while ValPtr <> nil do
                                                                                                                                             if switch
                                                                                                                                             NewVal := ValPtr;
procedure catenate(LeftArg, RightArg: TypeValTabPtr);
{
dyadic comma operator – joins 2 arguments
}
begin { catenate }
    NewValues := nil;
    ResultLength := 0;
    begin { transfer of values }
        if LeftArg. dimensions <> 1 then
            NewValues := new(NewValues);
            NewValues. RealVal := LeftArg. FirstValue;
            LeftValPtr := LeftArg. FirstValue;
            while LeftValPtr <> nil do
                begin { transfer left arg values (if any) }
                    NewValues := new(NewValues);
                    NewValues. RealVal := LeftValPtr. RealVal;
                    if LeftValPtr. RealVal = nil then
                        NewValues. RealVal := 0;
                    NewValues. NextVal := NewValues;
                    NewValTabLink := NewValues;
                    LeftValPtr := LeftValPtr. NextValue;
                end; { transfer left arg values (if any) }

        else
            if LeftArg. ForwardOrder = true then
                NewValues := new(NewValues);
                NewValues. RealVal := LeftArg. FirstValue;
                NewValues. NextVal := NewValues;
                NewValTabLink := NewValues;
                begin { transfer right arg values (if any) }
                    if RightArg. dimensions <> 1 then
                        NewValues := new(NewValues);
                        NewValues. RealVal := RightArg. FirstValue;
                        RightValPtr := RightArg. FirstValue;
                        while RightValPtr <> nil do
                            if NewValues. NextVal = nil then
                                NewValues := new(NewValues);
                                NewValues. RealVal := RightValPtr. RealVal;
                                RightValPtr := RightValPtr. NextValue;
                            end { transfer right arg values (if any) }
                    end; { transfer right arg values (if any) }

        end; { transfer of values }
    end; { transfer of values }
end { catenate };
procedure reshape(leftArg, RightArg: TypeValTabPtr);

var

ResultLength, elements: integer;
DimPtr: DimInfo;
NewPtr: values;

begin [ reshape ]
if LeftArg .dimensions > 1 then
error(556) [ left argument not a vector or a scalar ]
end

procedure InnerProductCLeftArg, RightArg: TypeValTabPtr);

begin { inner product is matrix multiplication }

if not RightArg .forwardOrder then
ReverselinkList(RightArg)

if not LeftArg .forwardOrder then
ReverselinkList(LeftArg)

if LeftArg .dimensions <> RightArg .dimensions + 2 then
NewValTabLink .dimensions := 0;

if LeftArg .dimensions < 0 then
falseTrue := LastLeftDim := 0;

if LeftArg .firstDimen <> nil then

begin [ copy all but last of left arg dims into result ]
LeftSkip := 1; DimPtr := LeftArg .firstDimen;

while DimPtr <> nil do

ResultLength := ResultLength * trunc(LeftValPtr .realVal);

NewDim := new(NewDim);

NewDim .dimensions := DimPtr .nextDimen .dimensions - 2;

NewDim .firstDimen := NewDim .nextDimen := 0;

DimPtr := DimPtr .nextDimen;

end

else NewPtr .nextDimen := NewDim;

NewPtr := NewValTabLink .firstDimen := NewDim .nextDimen := 0;

LastLeftDim := DimPtr .dimensions - 1;

end;

end { reshape }
procedure OuterProduce(LeftArg, RightArg: TypeValTabPtr);

var

OutProCode: integer;
Sfloat: real;

begin

OutProCode := code div 10; newValTabLink;

OldValTabLink.NextValTabLink := newValTabLink;

NewValTabLink := newValTabLink;

if not LeftArg.ForwardOrder
if switch then ReverseListLink(RightArg);

if not LeftArg.ForwardOrder

NewVal := newValTabLink.FirstValue;

if switch then

LeftValPtr := LeftArg.FirstValue;

if LeftValPtr <> nil do
begin

NewVal := newValTabLink.FirstValue;

while LeftValPtr <> nil do

begin

switch := false; NewValTabLink.FirstValue := New

end;

NewPtr := NewVal;

DimPtr := DimPtr.NextDimPtr;

end;

RightVal := RightArg.FirstValue;

while DimPtr <> nil do

begin

if switch then NewValTabLink.FirstValue := New;

else NewPtr.NextDimPtr := New;

NewPtr := NewVal;

DimPtr := DimPtr.NextDimPtr;

end;

switch := false; NewValTabLink.FirstValue := New;

LeftValPtr := LeftArg.FirstValue;

while LeftValPtr <> nil do

begin

RightVal := RightArg.FirstValue;

if switch then

begin

switch := false; NewValTabLink.FirstValue := New;

else NewPtr.NextDimPtr := New;

NewPtr := NewVal;

DimPtr := DimPtr.NextDimPtr;

end;

if switch then NewValTabLink.FirstValue := New;

else NewPtr.NextDimPtr := New;

NewPtr := NewVal;

DimPtr := DimPtr.NextDimPtr;

end;

switch := false; NewValTabLink.FirstValue := New;

LeftValPtr := LeftArg.FirstValue;

while LeftValPtr <> nil do

begin

RightVal := RightArg.FirstValue;

if switch then

begin

switch := false; NewValTabLink.FirstValue := New;

else NewPtr.NextDimPtr := New;

NewPtr := NewVal;

DimPtr := DimPtr.NextDimPtr;

end;

if switch then NewValTabLink.FirstValue := New;

else NewPtr.NextDimPtr := New;

NewPtr := NewVal;

DimPtr := DimPtr.NextDimPtr;

end;

if switch then

begin

switch := false; NewValTabLink.FirstValue := New;

end;
procedure funcall(var ValidFunk: Boolean);

begin (funcall)
  ValidFunk := false;
  if funcall then
    begin (dyadic)
      if TokenTabPtr.noun = $END then
        begin (end)
          if TokenTabPtr, TokenCallingSubr := TokenTabPtr;
        end;
      else begin (primaryValidFunk)
        if not ValidFunk then error(17); 
      end;
      (leftarg of dyadic func not a primary)
    end;
  end (funcall)

end;
Software Tools

```
2801 then
2802 begin
2803 NewFuncTabPtr->NextFuncTabPtr = OldFuncTabPtr;
2804 OldFuncTabPtr = NewFuncTabPtr;
2805 NewVFuncPtr->NextVFuncPtr = nil;
2806 end
2807 else
2808 if Error(75) (function defined with no statements) then
2809 functionMode := false; position := position + 1
2810 end
2811 else
2812 ProcessFunctionHeader (start of a new function)
2813 else not a del encountered )
2814 begin
2815 if TokenSwitch then
2816 begin
2817 (this is start of a new statement)
2818 TokenSwitch := false;
2819 HasTokenPtr := NewTokenPtr (save starting position)
2820 MakeTokenLabel.NewTokenPtr->noun := StatEnd;
2821 NewTokenPtr->EndPos := 0;
2822 HasLabel := false
2823 NewTokenLabel := identifier(name, ItsAnIdentifier);
2824 if not ItsAnIdentifier then TryToGetANumber
2825 begin (process identifier)
2826 skipSpaces;
2827 if (APLstatement(position) = character(colon) ) and
2828 (NewTokenPtr->NextToken->noun = StatEnd)
2829 then
2830 begin (process statement label)
2831 SaveLabel := name; HasLabel := true; position := position + 1
2832 end
2833 else
2834 begin (process variable name)
2835 if not functionMode then NewTokenPtr->noun := GlobVar
2836 else
2837 if NamesMatch(name, NewFuncTabPtr->ResultName)
2838 then NewTokenPtr->noun := FormRes
2839 else if (NamesMatch(name, NewFuncTabPtr->LeftArg) or
2840 (NamesMatch(name, NewFuncTabPtr->RightArg))
2841 then NewTokenPtr->noun := FormArg
2842 else NewTokenPtr->noun := GlobVar;
2843 if HasTokenPtr->noun = GlobVar
2844 then TestFuncPtr = NewTabPtr
2845 else TestFuncPtr = nil;
2846 if not NameInVarTable(name, VarPointer, TestFuncPtr)
2847 then
2848 begin (add name to variable table)
2849 NewTokenPtr->VarTabPtr := NewVarTabPtr
2850 else
2851 NewTokenPtr->VarTabPtr := VarPointer
2852 end
2853 end
2854 end
2855 end
2856 end; skilSpaces;
2857 if NewTokenPtr <> nil then
2858 if (TokenError) or (NewTokenPtr->noun = StatEnd)
2859 then DestroyStatement
2860 else functionMode
2861 then
2862 begin (begin a function statement)
2863 functionStatements := functionStatements + 1;
2864 if functionStatements > 0
2865 then
2866 begin (catalog function statement)
2867 newNewFuncPtr;
2868 if functionStatements = 1
2869 then NewFuncPtr->FirstStatement := NewVFuncPtr;
2870 else OldVFuncPtr->NextVFuncPtr := NewVFuncPtr;
2871 OldVFuncPtr = NewVFuncPtr;
2872 if HasLabel
2873 then NewVFuncPtr->StatLabel := SaveLabel;
2874 NewVFuncPtr->NextStmt := NewTokenPtr
2875 end
2876 else
2877 if APLstatement(i) <> character(del) then
2878 begin
2879 if NameInVarTable(NewTokenPtr, NewVarTabLink);
2880 100: DestroyStatement;
2881 reading: TokenError := false; GetAPLStatement;
2882 end
2883 end
2884 end; scanner.

Contents of APLFile

```

---
The government imposed a 55-mph speed limit on cars, not computers. Why, then, are computer owners going so slowly?

Are we in the early stages of a technology bust? Strange as this may sound at a time when the nation seems to have gone computer crazy, a good many scientists are starting to worry about just that.

Their concern stems from the massive switch in the computer business from a customer base consisting of a handful of large institutional buyers to millions of smaller ones. The computer finally has become a piece of mass-market electronics, much like video recorders. Why is that a problem? A basic rule of business is that risk accompanies opportunity. In this instance, the risk affects not only the companies in the field, but the entire country, thanks to the expanding economic importance of this industry. Its health will soon playa decisive role in determining the U.S.' international competitive position.

The risk in dealing with the mass market is always a simple one: The mob is fickle. What intrigues it today may leave it indifferent tomorrow. This time the fickleness could produce a national disaster. The U.S. has unwittingly invested a major portion of its capital — and even more important, its hopes — in this area. That’s why some thoughtful workers in the field are beginning to pray quietly: “Don’t fail me now.”

How, specifically, do they see a failure occurring? The consensus view is as follows: “A Ferrari is exciting, but how exciting would it continue to be if the only place you could use it were your driveway? Well, that’s exactly what is happening with too many of the computers now being bought. Car or computer, people are eventually going to get tired of just looking at the thing and bragging about it to their friends. Then, the fad will pass. Computer manufacturing plants will close. Only a minuscule proportion of computer buyers are making good use of the machine’s capabilities. They don’t know enough about programming to make the machine really perform.”

“ Well, suppose everyone learned BASIC?” I asked.

The overwhelming majority had a better idea: “BASIC is a very easy language to learn, but it would be enormously better, a dream come true, if everyone learned Pascal, which is far superior and just as easy to master.”

Since last summer I, therefore, have been collecting the opinions of everyone, from teachers and hobbyists to investors and small business owners, who know Pascal to see which books they consider best. A tally of the nearly 1,600 replies shows the following:

For people who know nothing at all about computers or computer programming, the best place to begin is R. Pattis’ Karel the Robot: A Gentle Introduction to the Art of Programming (John Wiley, $8.95). You don’t need a computer to read this book (or the others about to be mentioned). By learning how to move a robot through the streets of a small town, you come to understand how programming instructs a computer to do what you want it to.

Pattis’ book is about programming but doesn’t actually teach the language. The elementary text that received the top rating in our survey was Arthur Keller’s A First Course in Computer Programming with Pascal (McGraw-Hill, $14.95). The book received high praise (“Very clear and easy to read”) from everyone from 17 to 70. It is suitable even as a high school text.

After Keller’s book, the next step should be A Primer on Pascal by Conway, Gries and Zimmerman (Little, Brown, $20). The consensus view: “This book will help you deepen your understanding of the language once you’ve learned the elements.” For those who already know BASIC, a good way to learn Pascal fast is Quick Pascal by D. Matuszek (John Wiley, $11.95).

One work that was highly rated by advanced students was the second edition of Pascal — User’s Manual and Report (Springer-Verlag, $10.50) by K. Jensen and N. Wirth. That is hardly surprising since one of the coauthors, Nikolaus Wirth, invented the language.

To see what the language can really do, serious students will want to learn about data structures — that is, such things as lists, stacks, queues, trees, sets, records, recursion, sorting and searching. The three top-rated texts, all very well written, are: Data Structures and Algorithms by A. Aho, et al. (Addison-Wesley, $28.95); Advanced Programming and Problem Solving with Pascal by G. Schneider and S. Bruehl (John Wiley, $26.95); and Data Structures Using Pascal by A. Teenenbaum and M. Augenstein (Prentis-Hall, $25.95). As the authors of the first work comment, “The only prerequisite we assume is familiarity with some high-level programming language such as Pascal.”

Finally, people with a background in probability theory rated the second edition of R. Cooper’s Introduction to Queuing Theory (North-Holland Publishing Co., $27) the best — clearest and most user-friendly — book on the subject.

Summing up: Buying a computer and not learning to program it properly not only wastes money, it also stands a good chance of eventually harming the nation’s economy.

Dr. Srully Blotnick is a research psychologist and author of Getting Rich Your Own Way and Winning: The Psychology of Successful Investing.

Reprinted by permission of Forbes magazine, February 28, 1983.
Background

During the past twenty years the development of computer applications in geography has been extensive. Hundreds of programs have been written and many are available to users through various dispensing institutions, particularly the Geography Program Exchange located at Michigan State University. Virtually all of the programs, however, are written in FORTRAN and are suitable for easy installation primarily on large mainframe computers.

A similar situation exists in cartographic computer program development. Although the number and variety of programs written is extensive, the FORTRAN language is used almost exclusively, and the software is designed for use on large systems. A recent textbook in computer-assisted cartography provides only passing mention of microcomputer graphics in the field of cartography.

As a consequence of this situation, computer applications in geography and cartography are limited primarily to the larger colleges and universities that have mainframes and the faculty within the departments to teach the subjects. As a geographer in a small liberal arts college teaching not only introductory cartography but a course in micro-based computer mapping, I feel somewhat like a pioneer trying to make a clearing in the wood without the proper tools. The situation is further exacerbated because liberal arts colleges have not been as highly revered by the computer industry as have the high technology learning centers and are consequently not receiving anywhere near the number of equipment grants or the same degree of personnel support.

Yet, more than one writer has commented on the need for a closer association between the computer industry and the liberal arts college. In a recent editorial in Datamation, John L. Kirkely stated the following:

We urge our industry to work with the liberal arts colleges to develop courses of study that combine the humanities and the sciences. A merging of these artificially separated disciplines could be a powerful tonic for both our colleges and our corporations.

I believe that, in time, changes will be made which will result in the liberal arts colleges receiving their fair share of industry support. In the meantime, however, individuals in those colleges will continue to make contributions to the furtherance of computer applications in what would be considered today to be non-traditional disciplines. The set of programs included in this paper are suggestive of the kinds of things faculty can produce and which 1) are effective vehicles for developing the understanding of key concepts in a discipline (demography in this case); 2) are produced with a cost factor reflecting only the programmer’s time; 3) can be easily implemented on any system, micro to mainframe; and 4) are written in the programming language of the day, Pascal.

The Population Pyramid (Age Structure) Diagram

It is abundantly clear that world population continues to grow at a less than acceptable rate, and that some regions, particularly those with countries exhibiting low levels of economic development, have exceptionally high rates of growth. The population pyramid is a useful diagram to study the composition of the population of any country or region.

In the diagram, age groupings of five years each (0-4, 5-9, 10-14,..., up to 75+) are presented for both male and female segments of the population. The scale along the horizontal axis reports the percentage of the total population in each of the age groups. Generally a pyramid shape wide at the bottom (young age groups) is representative of a fast growing population while an age structure more evenly represented along the year's axis identifies a population that is stabilizing and that does not have a high rate of increase. The industrialized and urbanized countries in the developed world would fall into the latter category; the less developed in the former.

The Programs

Three programs have been developed for student use in an introductory human geography course. It is not necessary that the students know the Pascal language in order to run the programs. Introduction is given in class on login/logoff procedures and how to access the programs. The student need only find suitable information in the appropriate statistical source for each of the age groups for a particular region, round these values to a whole number, and enter the numbers in the program prompts. The programs developed include:

1) pyramid_file — This program is used to create an external file of information including the region names, year of the data, and the percentages of male and female in each age group. Following the input, procedure echo-data publishes all the information entered for verification. If there were no input errors, the student selects the appropriate key and the program stores the information in an external file in the student’s account. The listing of program pyramid_file follows:
program pyrdata(input, output):
(a program to create an external file of
population pyrdata data)

const
separator = ' ';

type

data =

record

countries packed array (1,10) of chars;
years packed array (1,6) of chars;
relpercent, female relpercent array (1,10) of integers;
identifiers = file of data;

var
temp data;
infos identifiers;
filenames packed array (1,10) of chars;
countries

procedure read_data:

var

i: integer;

begin

writeln(' enter file name: ');
readln(filenames);
writeln(' enter place name = use 16 columns: ');
readln(temp,countries);
writeln(' enter the year of the data: ');
readln(temp,years);
writeln(' now, enter male and female percentages: ');
readln(temp,relpercent);
writeln(' from the highest age group to the lowest: ');
readln(temp,female relpercent);
writeln(' enter as integers all on the same line: ');
for i = 1 to 10 do

read(temp,relpercent);
writeln(' next, enter the female percentages: ');
for i = 1 to 10 do

read(temp,female relpercent);
end (read_data);

procedure echo_data:

var

i: integer;

begin

writeln(separator);
writeln(' the following information was entered: ');
writeln(' ', temp, countries);
writeln(' ', temp, years);
writeln(' male: ');
for i = 1 to 10 do

writeln(temp,relpercent);
writeln(' female: ');
for i = 1 to 10 do

writeln(temp,female relpercent);
writeln(separator);

end (echo_data);

procedure store_data:

var

i: integer;

begin

rewrite(info, filenames);
info, country = temp, country;
put(info);
info, year = temp, year;
put(info);
for i = 1 to 10 do

begin
info, relpercent[i] = temp, relpercent[i];
put(info);
end;
for i = 1 to 10 do

begin
info, female relpercent[i] = temp, female relpercent[i];
put(info);
end;
close(info);
end (store_data);
end (read_data);

procedure skiplines:

var

i: integer;

begin
for i = 1 to 10 do

write('');
end (skiplines);

procedure accessfile_data:

var

i: integer;

begin
writeln(separator);
writeln(' the following information is: ');
writeln(' ', temp, country);
writeln(' contained in ', filenames,' ');
write(info);
info, country = temp, country;
put(info);
info, year = temp, year;
write(info);
write(info, male relpercent);
for i = 1 to 10 do

begin
write(temp, rect[i]);
get(info);
end;
close(info);
end (accessfile_data);

procedure publish_data:

const

separator = ' ';

var

i: integer;

begin

skiplines;
write(separator);
writeln(' the following information is: ');
writeln(' contained in ', filenames,' ');
write(info);
write(info, country);
write(info, year);
write(info, male relpercent);
for i = 1 to 10 do

begin
write(temp, rect[i]);
end;
close(info);
write(info);
end (publish_data);

begin

read_date;

if (answer = 'y') or (answer = 'Y') then

begin
store_date:

write(' operation completed ');
write(' information stored in the file: ');
end;
else

begin
write(' invalid data; run the program again');
end;

2) get_pyr_file — An editing program which the student may use to access an external file, display the contents, and make any necessary changes. This program would come in handy if more recent data is received and the file is to be updated. The listing or program get_pyr_file and an example run of information contained in the external file, SWEDEN.PYR, follow:

program get_pyr_file(input, output):
(a program to examine the external file created by the program 'pyrdata_file', and to make changes if necessary)

var
temp data;
infos identifiers;
filenames packed array (1,10) of chars;

begin

writeln;
write(' 

follow: ');

read_dlltlll;
writeln;
writeln(' enter file name: ');
readln(filenames);
writeln(' enter place name: use 16 columns: ');
readln(temp,countries);
writeln(' enter the year of the data: ');
readln(temp,years);
writeln(' now, enter male and female percentages: ');
readln(temp,relpercent);
writeln(' from the highest age group to the lowest: ');
readln(temp,female relpercent);
writeln(' enter as integers all on the same line: ');
for i = 1 to 10 do

read(temp,relpercent);
writeln(' next, enter the female percentages: ');
for i = 1 to 10 do

read(temp,female relpercent);
end (read_data);

begin

writeln(separator);
writeln(' the following information was entered: ');
writeln(' ', temp, countries);
writeln(' ', temp, years);
writeln(' male: ');
for i = 1 to 10 do

writeln(temp,relpercent);
writeln(' female: ');
for i = 1 to 10 do

writeln(temp,female relpercent);
writeln(separator);

end (echo_data);

begin

rewrite(info, filenames);
info, country = temp, country;
put(info);
info, year = temp, year;
put(info);
for i = 1 to 10 do

begin
info, relpercent[i] = temp, relpercent[i];
put(info);
end;
for i = 1 to 10 do

begin
info, female relpercent[i] = temp, female relpercent[i];
put(info);
end;
close(info);
end (store_data);

begin

writeln;
write(' 

operation completed ');
write(' information stored in the file: ');
end;

begin
write(' invalid data; run the program again');
end;

end (get_pyr_file);

begin

writeln;
write(' 

follow: ');

read_dlltlll;
writeln;
writeln(' enter file name: ');
readln(filenames);
writeln(' enter place name: use 16 columns: ');
readln(temp,countries);
writeln(' enter the year of the data: ');
readln(temp,years);
writeln(' now, enter male and female percentages: ');
readln(temp,relpercent);
writeln(' from the highest age group to the lowest: ');
readln(temp,female relpercent);
writeln(' enter as integers all on the same line: ');
for i = 1 to 10 do

read(temp,relpercent);
writeln(' next, enter the female percentages: ');
for i = 1 to 10 do

read(temp,female relpercent);
end (read_data);

begin

writeln(separator);
writeln(' the following information was entered: ');
writeln(' ', temp, countries);
writeln(' ', temp, years);
writeln(' male: ');
for i = 1 to 10 do

writeln(temp,relpercent);
writeln(' female: ');
for i = 1 to 10 do

writeln(temp,female relpercent);
writeln(separator);

end (echo_data);

begin

rewrite(info, filenames);
info, country = temp, country;
put(info);
info, year = temp, year;
put(info);
for i = 1 to 10 do

begin
info, relpercent[i] = temp, relpercent[i];
put(info);
end;
for i = 1 to 10 do

begin
info, female relpercent[i] = temp, female relpercent[i];
put(info);
end;
close(info);
end (store_data);

begin

writeln;
write(' 

operation completed ');
write(' information stored in the file: ');
end;

begin
write(' invalid data; run the program again');
end;

end (get_pyr_file);

Articles
procedure make_file_changes;

    var
        i: integer;
        selector: char;
    begin
        writeln('you may make any number of changes');
        writeln('my selecting the appropriate selector');
        writeln('for the data to be changed.');
        writeln('use the following list of selectors');
        writeln('area name == "area"');
        writeln('year == "year"');
        writeln('male percent == "male"');
        writeln('female percent == "female"');
        writeln('enter the selector, then <cr>, and ";
        writeln('you will be prompted to enter the');
        writeln('new data.');
        writeln('when you have completed the changes');
        writeln('enter an "e" to end the session.');
        writeln('enter a selector:');
        readln(selector);
    repeat
        case selector of
            'a', 'A':
                begin
                    writeln('enter the new area name:');
                    readln(area_info, country);
                    write('year: '); readln(year);
                    for i := 1 to 10 do
                        readln(x[i], y[i]);
                    writeln('enter all six percents:');
                    for i := 1 to 4 do
                        readln(x[i], y[i]);
                end;
            'r':
                begin
                    writeln('enter four years:');
                    readln(year);
                    writeln('enter all six percents:');
                    for i := 1 to 10 do
                        readln(x[i], y[i]);
                end;
            'e':
                begin
                    writeln('enter new data stored in "", file_name');
                    writeln('publish data end (make file changes)');
    procedure modify_file_changes;

    var
        choices: char;
    begin
        writeln('do you want to modify the data ?
        if yes, enter a "y"');
        writeln('if no, enter a "n"');
        readln(choices);
        if choices = 'y' then
            modify_file_changes;
        if choices = 'n' then
            end (modify_file_changes);
    end (modify_file_changes);

    3) drawpyramid — The final program accesses the
    information stored in the external file and produces a
    pseudo-graphic on a line printer. The program can
    produce a single plot, as shown in the BERLIN example,
    or a double plot of either one region in two time periods
    or two different regions. The student selects single or
    double plot and enters the file names. The program takes
    over from here and produces the output. A double plot
    of SWEDEN and MEXICO illustrates the age structures
    of a country with a low rate of growth and one
    which is high.

    run getpyr
    enter file name: sweden.py

    sweden
    1970
    male percent: 1 2 2 3 3 4 4 4 3 3 4 4
    female percent: 1 2 2 3 3 4 4 4 3 3 4 4

    do you want to modify the data ?
    if yes, enter a "y";
    if no, enter an "n": n
    no changes to the file.
    Ready

    program drawpyramid(input, output);

    (a program to produce a population pyramid graphic)
    const
        blank = ' ';
    type
        data = record
            country packed array [1..15] of char;
            years packed array [1..10] of char;
            male, female: array [1..10] of integer;
        end;
        identifiers = file of data;
        filename = packed array [1..10] of char;
    var
        choices char;
        temp data:
            matrix: array [1..42, 1..63] of char;
            pyramid: identifiers;
            file1, file2: filename;
        procedure initialize_arrays (set all array elements to blank)
        var
            i, j: integer;
        begin
            for i := 1 to 42 do
                for j := 1 to 63 do matrix[i, j] := blank;
            file1 := pyramid;
            file2 := pyramid;
            for i := 1 to 15 do
                matrix[1, i] := blank;
            for i := 1 to 42 do
                for j := 1 to 63 do
                    if matrix[i, j] = blank then
                        matrix[i, j] := blank;
            for i := 1 to 15 do
                matrix[i, 1] := blank;
            for i := 1 to 42 do
                for j := 1 to 63 do
                    if matrix[i, j] = blank then
                        matrix[i, j] := blank;
            file1 := file1 + file1;
            file2 := file2 + file2;
            for i := 1 to 15 do
                matrix[i, 1] := blank;
            for i := 1 to 42 do
                for j := 1 to 63 do
                    if matrix[i, j] = blank then
                        matrix[i, j] := blank;
            file1 := file1 + file1;
            file2 := file2 + file2;
            for i := 1 to 15 do
                matrix[i, 1] := blank;
            for i := 1 to 42 do
                for j := 1 to 63 do
                    if matrix[i, j] = blank then
                        matrix[i, j] := blank;
            file1 := file1 + file1;
            file2 := file2 + file2;
            for i := 1 to 15 do
                matrix[i, 1] := blank;
            for i := 1 to 42 do
                for j := 1 to 63 do
                    if matrix[i, j] = blank then
                        matrix[i, j] := blank;
        procedure plot(choices: (single plot or superimposed plot)
        begin
            writeln('if choice is "d" it is a double plot
            if choice is "s" it is a single plot
            readln(choices);
            end (plot choices)

            procedure enter_file_name;
        begin
            if choice is "d" or choice is "s" then double plot
            begin
                writeln('enter each file name on a separate line');
                writeln('use ten columns for each file name
                if you are at the line printer');
                writeln('last line of the paper before');
                writeln('entering <cr> after the second file name');
                writeln('write **>>**');
                for i := 1 to 15 do
                    writeln('');
            end (enter file name)

    3) drawpyramid — The final program accesses the
    information stored in the external file and produces a
    pseudo-graphic on a line printer. The program can
    produce a single plot, as shown in the BERLIN example,
    or a double plot of either one region in two time periods
    or two different regions. The student selects single or
double plot and enters the file names. The program takes
over from here and produces the output. A double plot
of SWEDEN and MEXICO illustrates the age structures
of a country with a low rate of growth and one
which is high.

run getpyr
enter file name: sweden.py

sweden
1970
male percent: 1 2 2 3 3 4 4 4 3 3 4 4
female percent: 1 2 2 3 3 4 4 4 3 3 4 4

do you want to modify the data?
if yes, enter a "y";
if no, enter an "n": n
no changes to the file.
Ready

program drawpyramid(input, output);

(a program to produce a population pyramid graphic)
const
blank = ' ';
type
    data = record
        country packed array [1..15] of char;
        years packed array [1..10] of char;
        male, female: array [1..10] of integer;
    end;
    identifiers = file of data;
    filename = packed array [1..10] of char;
var
    choices char;
    temp data:
        matrix: array [1..42, 1..63] of char;
        pyramid: identifiers;
        file1, file2: filename;
procedure initialize_arrays (set all array elements to blank)
begin
    for i := 1 to 42 do
        for j := 1 to 63 do matrix[i, j] := blank;
    file1 := pyramid;
    file2 := pyramid;
    for i := 1 to 15 do
        matrix[1, i] := blank;
    for i := 1 to 42 do
        for j := 1 to 63 do
            if matrix[i, j] = blank then
                matrix[i, j] := blank;
    file1 := file1 + file1;
    file2 := file2 + file2;
    for i := 1 to 15 do
        matrix[i, 1] := blank;
    for i := 1 to 42 do
        for j := 1 to 63 do
            if matrix[i, j] = blank then
                matrix[i, j] := blank;
    file1 := file1 + file1;
    file2 := file2 + file2;
    for i := 1 to 15 do
        matrix[i, 1] := blank;
    for i := 1 to 42 do
        for j := 1 to 63 do
            if matrix[i, j] = blank then
                matrix[i, j] := blank;
    file1 := file1 + file1;
    file2 := file2 + file2;
    for i := 1 to 15 do
        matrix[i, 1] := blank;
    for i := 1 to 42 do
        for j := 1 to 63 do
            if matrix[i, j] = blank then
                matrix[i, j] := blank;
end (initialize_arrays)

procedure plot(choices: (single plot or superimposed plot)
begin
    writeln('if choice is "d" it is a double plot
    if choice is "s" it is a single plot
    readln(choices);
end (plot choices)

procedure enter_file_name;
begin
    if choice is "d" or choice is "s" then double plot
    begin
        writeln('enter each file name on a separate line');
        writeln('use ten columns for each file name
        if you are at the line printer');
        writeln('last line of the paper before');
        writeln('entering <cr> after the second file name');
        writeln('write **>>**');
        for i := 1 to 15 do
            writeln('');
end (enter file name)
end if
end

begin
if (choice = '0') or (choice = 'n') then (double plot)
begin
filearray[1] = file1;
filearray[2] = file1;
for i = 1 to 2 do
begin
if i = 1 then
k = 43
else
k = 41
reset(filearray[i], filearray[i]);
temp,country = yearid,country;
get(papyrus);
if k = 2 then
for j = 1 to 4 do
matrix[5, j + 1] = temp,country[j];
else
for j = 1 to 4 do
matrix[5, j + 1] = temp,year[j];
end
if k = 16 do
begin
temp,year = yearid,years;
get(papyrus);
matrix[2] = temp,year[1];
matrix[2] = temp,year[2];
matrix[2] = temp,year[3];
matrix[2] = temp,year[4];
end
end
close(papyrus);
end
else (single plot)
begin
reset(filearray1, filearray1);
temp,country = yearid,country;
get(papyrus);
for j = 1 to 4 do
matrix[1, j + 1] = temp,country[j];
end
if k = 16 do
begin
temp,year = yearid,years;
get(papyrus);
matrix[2] = temp,year[1];
matrix[2] = temp,year[2];
matrix[2] = temp,year[3];
matrix[2] = temp,year[4];
end
end
close(papyrus);
end

begin
procedure symbol, explanation;
var
check1, check2, data:
end
begin
write('Enter the file name using ten columns:');
write('Note the format of the filename');
write('If you are at the line printer:');
position the printing rear to the line
print the last line of the paper before:
end
begin
write('Enter the choice:');
begin
end
end

procedure retrieve end exercised:
begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end

begin

end

end
Conclusion

The inclusion of exercises such as this one in social science classes has proven to be valuable in a number of ways. It allows students with little or no programming background to get over their tentativeness about approaching a computer. In addition, I believe that such exposure to computers, however limited, contributes to the overall computer literacy of students. Finally, the experience may spur a student to want to take a course in computer programming or to learn other uses of the computer.

There is absolutely no reason why students in all divisions of the liberal arts setting should not benefit by the opportunities available in the field of computer science.

Notes

1. The Geography Program Exchange assists universities and other non-profit organizations with the interchange of computer software which relates to problems of a geographic nature. The address is:

   Geography Program Exchange
   Department of Geography
   Michigan State University
   East Lansing, Michigan 45824

2. One of only a few books written on the general topic of computer applications in geography is Paul M. Mather, Computers in Geography: A Practical Approach (Oxford: Basic Blackwell, 1976); it contains four chapters, one of which is an introduction to the FORTRAN language.


6. The programs in this paper were prepared using Oregon Software Pascal, Version 2.0, and run on a DEC PDP 11/70.
Path Pascal
A Language for Concurrent Algorithms

By W. Joseph Berman
Advanced Programming Techniques, Inc.
704 Village Road, Charlottesville, VA 22903

1. Introduction

This paper is intended to provide an overview of the Path Pascal programming language. Rather than introduce the language by studying its definition, the approach taken here is to explore a moderately complex example. While any detailed understanding of Path Pascal must be based upon the formal definition of the language, this paper will present the most important concepts embodied in the language.

After a brief history of the development of Path Pascal, the problem to be solved by the example program will be presented. Using this example, three major concepts of Path Pascal will be explored. With these major concepts, the operation of the program can be understood. Finally, a summary of the present status and anticipated future of Path Pascal are discussed.

2. Background

Path Pascal was originally developed at the University of Illinois in 1978 by Dr. Roy Campbell. Details of the original Path Pascal compiler project are available from the University of Illinois as a series of Research Reports. In addition, the definition of the language has appeared in SIGPLAN Notices [1].

The University of Illinois implementation of Path Pascal was for the LSI-11/23 processor. Path Pascal has now been implemented on a variety of machines including the M68000 (by NASA-LaRC), AMAC-80 (by Martin-Marietta) and VAX-11/780 (by NASA-Goddard). This report is based upon experience gained with Path Pascal as part of NASA Contract NASI-16985 during 1982 using the M68000-based Path Pascal compiler developed by Dr. Ed Foudriat at NASA’s Langley Research Center.

Path Pascal is based upon “Path Expressions” first described by Campbell and Habermann in 1974[2]. The key concept is that coordination among a collection of concurrent processes should be expressed in a language designed especially for that purpose. “Path Expressions” conveniently and succinctly specify the central concepts of “mutual exclusion” (protecting “critical sections” of code) and of “synchronization” (waiting for information to be computed by other processes). In Path Pascal, the primary unit for mutual exclusion and synchronization is the subroutine, allowing the use of symbolic names in the Path Expressions.

In Path Pascal, “counting semaphores” are used to implement both mutual exclusion and synchronization. By specifying the Path Expression prior to the subroutines that it controls, the compiler can generate appropriate initialization, P-operation and V-operation at the beginning and ending of each subroutine.

2.1 The Island

Before discussing the example Path Pascal program, it may be useful to understand that this program is a simple event-driven simulation. The simulation involves an island and its inhabitants.

The island of this program is a very special island. It consists of a 25 × 17 grid, each element of which can either be empty (displayed as a blank), contain a wolf (designated by a ‘W’), or contain a rabbit (designated by a ‘R’). Initially, there are 17 wolves (all in column 10) and 17 rabbits (all in column 16).

Each wolf begins with a user-specified “energy”. This energy is used on an “annual” basis to remain alive, looking for rabbits to eat or other wolves with which to mate. Each “year” the wolf looks around his position on the grid, determining if there are any rabbits or wolves in his neighborhood. If there are any rabbits, the wolf’s energy is increased by eating them. If, on the other hand, there are too many wolves in the neighborhood, the wolf loses excess energy due to overcrowding. Only if there are a reasonable number of neighbor wolves and this wolf is “fertile” does the wolf attempt to produce an offspring. If the wolf’s energy is reduced to zero, it dies.

Each rabbit also begins with a user-specified “energy”. This energy is affected in a manner similar to a wolf, except that a rabbit is considered to have been “eaten” if there are any wolves in its immediate neighborhood and that rabbits gain energy by overcrowding rather than losing it.

Finally, the user may wish to “repopulate” the island, assigning new energy and lifetime specifications. This is done by pressing any key on the keyboard.

3. Path Pascal Constructs

The example program, ISLAND, is primarily written in standard Pascal. It makes use of only three new constructs—OBJECTs, PROCESSes and Wait-for-Son processing.

3.1. OBJECTs

Of the several extensions to standard Pascal, the OBJECT construct is the most important to understanding Path Pascal. An OBJECT is a Path Pascal TYPE with several properties similar to a RECORD. As with a RECORD, each variable of an OBJECT TYPE allocates stack space and NEW of a pointer to an OBJECT TYPE allocates heap space. The space required for a OBJECT TYPE is that for the semaphores implied by the OBJECT’s Path Expression and for any variables explicitly declared within the OBJECT. Unlike RECORDs, OBJECTs contain subroutines (PROCES-
DURES, FUNCTIONs and PROCESSES). Those subroutines that are included in the Path Expression are termed "ENTRY" routines and may be accessed from outside of the OBJECT using the RECORD-like notation "object.entry(parameters)" (e.g., line 436 or line 444).

Lines 16-52, 71-95, 96-183 and 186-200 indicate four common kinds of OBJECTs. These examples will be discussed in the following sections.

3.1.1. OBJECT CRTOBJ (lines 16-52)

This OBJECT is an example of an "interprocess buffer" OBJECT. Since processes run concurrently (see Section 4), they must be synchronized in order to transfer information. Unlike ADA in which processes must "rendezvous", Path Pascal facilitates the concept of "interprocess buffers" that contain data to be transferred from one process to another. This allows the "sending" process to continue execution after generating the information for the other process.

The Path Expression on lines 19-21 has both mutual exclusion and synchronization expressions. The first two expressions simply state that the operations PUSH and POP are atomic (only one PUSH at a time and only one POP at a time). The last expression specifies the synchronization between PUSH and POP. It states that a call to POP may not proceed until a call to PUSH has completed and, furthermore, at most CRTSZ calls to PUSH can be honored before at least one call to POP occurs.

This last implication of the Path Expression, that at most CRTSZ calls to PUSH can proceed without at least one call to POP, is the key to understanding the data structures (lines 24-27) and code (lines 29-51) of CRTOBJ. Since at most CRTSZ calls to PUSH can occur without a call to POP, all that is required is space for CRTSZ "messages". Since the "interprocess message" in this case is just a character, BUF is simply an ARRAY of CRTSZ characters. INPTR specifies where PUSH is to put its character, and OUTPTR specifies where POP is to get its character. This code works because the Path Expression controls access to the routines PUSH and POP, and, therefore, controls access to the BUF ARRAY.

3.1.2. OBJECT CREATOR (lines 71-95)

This OBJECT is also an interprocess buffer; however, the buffer has only a single entry (the VARs XX, YY and EE). This form of the interprocess buffer is very similar to the ADA "rendezvous".

3.1.3. OBJECT SCREEN (lines 96-183)

This OBJECT is used to control access to the INFO ARRAY, the inmemory representation of the island. The Path Expression on line 99 simply states that one and only one of the allowed operations may be progress at any given instant.

The routines in this OBJECT include SETUP (for reinitialization), KILL (for termination), LOOK (for examining the "neighborhood" of a wolf or rabbit), ASSIGN (for direct control of INFO), CHANGE (a test-and-set operation) and DONE (a set-and-test operation). The routine WRITES (update terminal screen) is available only within this OBJECT.

3.1.4. OBJECT SHUTUP (lines 186-200)

This OBJECT is used to synchronize the termination of the simulation. Since Path Pascal does not allow "preemptive termination" of process (see Section 4), care must be taken when writing processes that must eventually terminate.

This OBJECT acts essentially as a binary semaphore. The call to SHUTUP_WAIT on line 376 will cause the calling process (SHUTDOWN) to suspend operation until the call to SHUTUP_SIGNAL is made on line 364.

3.2. PROCESSES

The second major addition of Path Pascal to standard Pascal is the PROCESS. Conceptually, a PROCESS is a PROCEDURE that, after it is called, executes in parallel with its caller. In Path Pascal, all processes that are not waiting due to a Path Expression, a DOIO (see below) or a DELAY are competing for the hardware processor(s). Also note that each call to a PROCESS creates a new process, as in lines 311-317.

PROCESSses in Path Pascal can either be normal or INTERRUPT PROCESSses. An INTERRUPT PROCESS has two special attributes not associated with normal PROCESSses. The PRIORITY and VECTOR information are used to control the interrupt hardware such that the DOIO statement (lines 213-237) acts as a "wait-for-interrupt". In addition, as shown on lines 211, 216, 234, 239, it is sometimes necessary to enter "supervisor state" in order to access device controllers.

3.2.1. INTERRUPT PROCESS DLVJIN (lines 201-221)

This process is an "infinite loop", waiting for an interrupt from the terminal input hardware. When such an input occurs, the character is forwarded to the appropriate interprocess buffer.

3.2.2. INTERRUPT PROCESS DLVJOUT (lines 224-241)

This process is also an "infinite loop". It waits for a character to be placed into the appropriate interprocess buffer, transfer the character to the terminal, and waits for the completion interrupt.

3.2.3. PROCESS WOLF (lines 242-270)

This process corresponds to a wolf in the simulation; it is called when a wolf is to be created. The process is a loop corresponding to the lifetime of the wolf. In the loop, the ENERGY of the wolf (a local variable) is constantly updated until it is reduced to zero and the wolf dies.

3.2.4. PROCESS RABBIT (lines 273-302)

This PROCESS is similar to PROCESS WOLF, except that it corresponds to a rabbit. As with the wolf, the ENERGY of the rabbit is constantly updated until it is reduced to zero and the rabbit dies.
3.2.5. PROCESS SHUTDOWN (lines 366-380)

This PROCESS is used to wait for input from the user (any input will do). When this input occurs, it is necessary to notify the screen monitor (line 374) and the main program (line 375). However, at this point it is necessary to wait for the main program (actually, PROCEDURE PROCREATE) to complete its processing (line 376). Finally, SHUTDOWN "absorbs" any extra attempts to create rabbits or wolves. This is completed when the special message having an ENERGY of zero is encountered, and SHUTDOWN is terminated.

3.3. Wait-for-Sons Processing

When a PROGRAM, PROCEDURE, FUNCTION or PROCESS calls a PROCESS, it is necessary that this "son" process terminate before the "father" can terminate. This is logically necessary due to the scope rules of Path Pascal. Furthermore, this "wait-for-sons" processing is a useful tool for coordinating the termination of a system.

Except for "wait-for-sons" processing, there is no reason that the code in PROCEDURE PROCREATE could not be part of the main program. Note, however, that all of the WOLF and RABBIT processes are initiated by PROCREATE. Hence, PROCREATE cannot continue until all of these processes have terminated. This fact is critical to the coordination between PROCREATE and SHUTDOWN when the simulation is being terminated.

4. PROGRAM ISLAND

Having looked at the special features of Path Pascal that are used by this program, it is now possible to step through a typical execution of the program.

The main program begins (lines 430-433) by allocating heap-space for CRTIBUF and CRTOBUF, by initiating the input/output processes DLJVIN and DLJVOUT, and associating DLJVIN with CRTIBUF and DLJVOUT with CRTOBUF.

The driving loop of the program (lines 435-445) clears the terminal’s screen (using DEC-VT52 protocol), prompts the user for parameters (PARAMS), reinitializes the simulation (SCREEN.SETUP), initiates a process to look for terminal input (SHUTDOWN), and calls PROCEDURE PROCREATE.

PROCEDURE PROCREATE (lines 303-365) begins by initiating 17 wolves and 17 rabbits. It then enters a loop waiting for requests for creation. When such a request occurs, it is first tested to see if it was generated by SHUTDOWN, indicating that termination should begin. If this is not a SHUTDOWN request, it is a request for the creation of a wolf (ENERGY>0) or a rabbit (ENERGY<0). Each direction (UP, DOWN, LEFT and RIGHT) is tested to see if it is available. If all directions are occupied, creation is not possible. If a free position on the island is found, the SCREEN.CHANGE call updates the simulation and the rabbit or wolf is created (lines 358-361). When the special SHUTDOWN request is encountered, PROCREATE signals SHUTDOWN that it has completed processing, and waits for all of the wolf and rabbit processes to terminate.

Once all of the wolf and rabbit processes terminate, PROCREATE returns to the main program (line 444). The main program now signals the SHUTDOWN process that no more requests for creation will be generated, and the master control loop iterates.

5. Summary and Conclusions

While a single example cannot cover all of the constructs and uses of these constructs, the ISLAND program is representative of the important capabilities that Path Pascal has that are not found in standard Pascal. These capabilities include multiple processes (PROCESS), interprocess coordination (OBJECT), and process termination coordination (Wait-for-Son).

Having programmed in Path Pascal for several months, it is clear that these new capabilities are useful. Many of the PROCESSes and OBJECTs that have been written have been found to be highly reusable since they "encapsulate" and entire concept or function within the program. However, it is equally clear that these new capabilities do not "solve" the concurrent programming problem. Developing the Path Expressions is a tedious, error-prone undertaking. Nonetheless, once a Path Expression is finally "correct", it is usually clear to anyone reading the code exactly what will occur when the program is executed.

One of the goals of the current research with Path Pascal is to identify various "prototype" Path Expressions. The "interprocess buffer" is a good example. If a few such prototypes can be found to be sufficient for most situations, a "macro OBJECT" facility might be added to Path Pascal to make these prototypes readily available to the average programmer.
ENTRY PROCEDURE ASSIGN(X, Y, FUNCTION CHANGE());
ENTRY PROCEDURE STARTUP(VAR X, Y, F.: INTEGER);
ENTRY PROCEDURE KILL, LOOK ASSIGN.CHANGE, DONE;
ENTRY PROCEDURE TEST(ENERGY INTEGER);
ENTRY PROCEDURE WRITE(X, Y, INTEGER);
ENTRY PROCEDURE ASSIGN(X, Y, INTEGER);
ENTRY FUNCTION CHANGE(X, Y, INTEGER, BOOLEAN);
ENTRY FUNCTION WRITE(X, Y, INTEGER);
ENTRY FUNCTION DONE(X, Y, INTEGER, BOOLEAN);
BEGIN PROCEDURE PROCREATE ( INTEGER ENERGY;)
BEGIN
   ENERGY := ENERGY + 2; (* REST BASED UPON ENERGY *)
   IF ENERGY = 0 THEN (* AVOID BECOMING A WOLF *)
      ENERGY := 0; (* CANNOT USE EXCESS ENERGY *)
      IF ENERGY = MAX THEN (* REST BASED UPON ENERGY *)
         ENERGY := MAX;
      END;
   END;
   IF ENERGY < MAX THEN
      ENERGY := ENERGY;
   END;
   IF ENERGY = 0 THEN (* REST BASED UPON ENERGY *)
   END;
   RETURN;
END; (* PROCREATE *)

BEGIN
   CREATE (X,Y); (* CREATE A WOLF *)
   PROCESS RABBIT (* PROCESS RABBIT *)
   RETURN;
END; (* PROCREATE *)

BEGIN
   CREATE (X,Y,ENERGY); (* CREATE A WOLF *)
   PROCESS RABBIT (* PROCESS RABBIT *)
   RETURN;
END; (* PROCREATE *)

BEGIN
   CREATE (X,Y,ENERGY); (* CREATE A WOLF *)
   PROCESS RABBIT (* PROCESS RABBIT *)
   RETURN;
END; (* PROCREATE *)

BEGIN
   CREATE (X,Y,ENERGY); (* CREATE A WOLF *)
   PROCESS RABBIT (* PROCESS RABBIT *)
   RETURN;
END; (* PROCREATE *)

BEGIN
   CREATE (X,Y,ENERGY); (* CREATE A WOLF *)
   PROCESS RABBIT (* PROCESS RABBIT *)
   RETURN;
END; (* PROCREATE *)
An Introduction to Modula-2 for Pascal Programmers

By Lee Jacobson and Bebo White
Jacobson, White, & Associates
San Francisco, CA

THE BACKGROUND AND HISTORY OF MODULA-2

Modula-2 (like Pascal) was developed at the ETH-Zurich under the direction of Niklaus Wirth (Institut fur Informatik). Its development grew largely from a practical need for a general purpose, efficiently implementable systems programming language. The first production use of Modula-2 occurred in 1981. Dr. Wirth’s book, ‘Programming in Modula-2’ was published by Springer-Verlag in 1982.

It is virtually impossible to examine Modula-2 without recognizing its roots in Pascal. In its original design, Pascal was intended to be a language suitable for teaching programming as a systematic discipline based on certain fundamental concepts clearly and naturally reflected within it. These concepts were largely centered around stepwise refinement of problem solutions and structured programming.

Inasmuch as Pascal is basically an academic language, its widespread use for a variety of applications has clearly exceeded its design intention. Hence, many extensions to the original Pascal definition have been designed. Likewise, it has attracted as many critics as it has disciples.

Modula-2 has assumed all of the positive features of Pascal, and has attempted to address its commonly recognized shortcomings. The result is a structured, modular, portable, readable, efficient, machine independent, flexible language.

This paper will address the primary differences between Modula-2 and Pascal with particular emphasis on some of those features which the authors consider quite significant. Programming examples will be given in both Modula-2 and Pascal.

MODULA-2’S DIFFERENCES FROM PASCAL

The Role of Modules in Modula-2

Modules are the most important feature distinguishing Modula-2 from Pascal. Relying heavily upon the concepts of scope and block, modules address the problem, usually found in large programs, of separating visibility from existence. In block-structured languages, the range in which an object (e.g., a variable or procedure) is known is called the object’s scope, and therefore, defines its visibility. However, an object’s visibility also binds its existence, in that objects are created when the block in which they reside is entered and destroyed when the block is exited. It should be possible to declare variables that maintain their values, but are visible only in a few parts of a program. Concurrently, there is also a need for closer control of visibility. A procedure should not be able to access every object declared outside of it when it only needs to access a few of them.

Syntactically, modules closely resemble procedures, but they have different rules about visibility and the existence of their locally declared objects. Consider the following declarations:

```
PROCEDURE Outside;
VAR x,y,z: INTEGER;

MODULE Mod:
IMPORT x;
EXPORT a,b,c;
VAR a,b,c: INTEGER;
PROCEDURE P1;
BEGIN
  a := a + 1;
  x := x + 1;
END P1;
END Mod;

VAR a,b,c, INTEGER,
PROCEDURE Outside:
VAR x,y,z: INTEGER;

MODULE Mod:
IMPORT x;
EXPORT a,b,c;
VAR a,b,c: INTEGER;
PROCEDURE P1;
BEGIN
  a := a + 1;
  x := x + 1;
END P1;
END Mod;

VAR a,b,c, INTEGER;
PROCEDURE Outside:
VAR x,y,z: INTEGER;
```

The only syntactic difference between the module Mod and a normal Pascal procedure declaration are the reserved word beginning the declaration (MODULE instead of PROCEDURE) and the presence of IMPORT and EXPORT declarations following the module heading.

The semantic differences are more interesting. The objects declared within Mod (a, b, c) exist at the same time as the variables x, y, and z, and remain so long as Outside is active. The objects named in Mod’s IMPORT list are the only externally declared objects visible within Mod (x but not y or z). The objects declared in Mod’s EXPORT list are the only locally declared objects visible outside Mod. Thus, a and P1 are accessible from Outside, but b and c remain hidden inside Mod.

Specifically, a module can be thought of as a syntactically opaque wall protecting its enclosed objects, be they variables or procedures. The export list names identifiers defined inside the module that are also to be visible outside. The import list names the identifier defined outside the module that is visible inside. Generally, the rules for modules are:

1. Locally declared objects exist as long as the enclosing procedure remains activated;
2. Locally declared objects are visible inside the module and if they appear in the module’s export list, they are also visible outside;
3. Objects declared outside of the module are visible inside only if they appear in the module’s import list;

The following example demonstrates the essence of modularity:
The random number generator in these examples uses a seed variable to generate the next random number. Thus, the seed must maintain its value across function calls. The program on the right shows the classical Pascal solution. Notice that Seed’s declaration is at the top of the program, while its initialization is forced to the bottom. Two obvious disadvantages arise from the scattering of Seed across the face of the program:

1. Its occurrences become hard to find, especially in a large program;
2. It becomes accessible to every other procedure in the program even though it is used only by Random;

The example on the left demonstrates the usefulness of the module structure. The only object visible to the outside world is the procedure Random, while all objects pertaining to the random number generator are contained in one place. Note that the module RandomNumbers contains both declarations and a statement part. Module bodies are the (optional) procedures and the (optional) beginning of a module declaration. Although subject to the module’s restrictive visibility rules, module bodies conceptually belong to the enclosing procedure rather than the modules themselves. Therefore, module bodies are automatically executed when the enclosing procedure is called.

Acknowledgments

The random number generator in these examples uses a seed variable to generate the next random number. Thus, the seed must maintain its value across function calls. The program on the right shows the classical Pascal solution. Notice that Seed’s declaration is at the top of the program, while its initialization is forced to the bottom. Two obvious disadvantages arise from the scattering of Seed across the face of the program:

1. Its occurrences become hard to find, especially in a large program;
2. It becomes accessible to every other procedure in the program even though it is used only by Random;

The example on the left demonstrates the usefulness of the module structure. The only object visible to the outside world is the procedure Random, while all objects pertaining to the random number generator are contained in one place. Note that the module RandomNumbers contains both declarations and a statement part. Module bodies are the (optional) outermost statement parts of module declarations and serve to initialize a module’s variables. Although subject to the module’s restrictive visibility rules, module bodies conceptually belong to the enclosing procedure rather than the modules themselves. Therefore, module bodies are automatically executed when the enclosing procedure is called.

Relaxed Declaration Order

New Pascal users are often frustrated and confused by the enforced declaration and definition block structure required within the program skeleton. Despite the emphasis on modules, blocks still play an important part in Modula-2: implementation modules, program modules, internal modules, and procedures are all declared as blocks. Differences from Pascal include relaxed order of declarations, termination of all blocks by a procedure or module identifier, and the optional nature of block bodies.

Pascal imposes a strict order on the declaration of objects; within any given block, labels must be declared before constants, constants before types, and so on. Modula-2 eliminates this restriction — declarations can appear in any order. Programs containing a large number of declarations are easier to read and understand when related declarations are grouped together (regardless of their kind).

The following is an example of a relaxed declaration order:

```
MODULE Xlator;
CONST MaxSym = 1024;
TYPE SymBuffer = ARRAY[1..MaxSym] OF CHAR;
VAR SymBuff1, SymBuff2, SymBuffer;
CONST MaxCode = 512;
TYPE CodeBuffer = ARRAY[1..MaxCode] OF BYTE;
VAR CodeBuff: CodeBuffer;
END Xlator.
```

This example easily demonstrates how various related declarations may be placed together in a Modula-2 program, whereas in a Pascal program they may be scattered due to strict block ordering. Relaxed declaration order not only improves readability but enables a logical ordering which may be very important in large programs.

GOTO-less Programming In Modula-2

Inasmuch as structured programming is often equated with elimination of the use of unconditional transfers, Pascal was designed to de-emphasize use of the GOTO statement. Still the GOTO statement and the LABEL 'type' were supported to allow programming cases where the Pascal logical structures were insufficient. This meant that a GOTO statement was available for use in a situation which would otherwise have forced restructuring of the program logic.

For example, consider the following two program segments:

```
Remainder := Alpha MOD Beta;
WHILE Remainder <> 0 DO
  Alpha := Beta;
  Beta := Remainder;
  Remainder := Alpha MOD Beta;
END;
```

```
Remainder := Alpha MOD Beta;
IF Remainder <> 0 THEN
  Alpha := Beta;
  Beta := Remainder;
END;
```

The example on the left avoids use of a GOTO by duplicating an operation. The example on the right, while using GOTOS is actually more explicit.

Modula-2 does not support Pascal GOTO and LABEL. Instead it provides transfer mechanisms for uses under particular controlled circumstances. One of these mechanisms is the EXIT statement which permits premature exiting of a loop. The following is a program segment in Modula-2 performing the same operation:

```
LOOP
  Remainder := Alpha MOD Beta;
  IF Remainder <> 0 THEN EXIT;
  Alpha := Beta;
END;
```

This example also illustrates the Modula-2 LOOP construct which operates as a Loop-Forever structure. When the EXIT statement is executed, program control
will transfer to the statement following the END state-
ment which terminates the range of the LOOP.

Additional examples of unconditional transfers
supported by Modula-2 include the RETURN state-
ment which is used to prematurely exit a procedure,
and the HALT standard procedure which terminates
the current program.

Dynamic Array Parameters

Another important distinction between Modula-2
and Pascal involves the capability to declare dynamic
array parameters. Modula-2 allows formal parameter
types of the form:

ARRAY OF T

where T is an arbitrary base type. Note that the array
bounds are omitted defining a dynamic array type which
is compatible with all (one dimensional) arrays having
the same base type T.

The ramifications of this feature are widespread.
through it, procedures are able to pass to other proce-
dures (functions, etc.) arrays of unspecified size. (Index
checking is accomplished by means of a new standard
procedure HIGH).

Perhaps the most important way in which dynamic
array parameters may be used is in the area of string
processing. This feature lifts the rigid Pascal restriction
concerning the value assignment and comparison of
string variables. No longer is it necessary that opera-
tions may only be performed on strings which have the
same length.

Separate Compilation

Separate compilation is allowed by the Modula-2
compiler through the use of the compilation unit. Mod-
ula-2 programs are constructed from two kinds of com-
pilation units: program modules and library modules. Program modules are single compilation units and their
compiled forms constitute executable programs. They
are analogous to standard Pascal programs.

Library modules are a different animal and form
the basis for the Modula-2 library. They are divided into
a definition module and an implementation module. Def-
inition modules contain declarations of the objects which
are exported to other compilation units. Implementa-
tion modules contain the code implementing the library
module. Both always exist as a pair and are related by
being declared with the same module identifier.

To understand the rationale behind dividing a li-
brary module into separate definition and implementa-
tion modules, consider the design and development
of a large software system, such as an operating system.
The first step in designing such a system is to identify
major subsystems and design interfaces through which
the subsystems communicate. Once this is done, actual
development of the subsystems can proceed, with each
programmer responsible for developing one (or more)
of the subsystems.

Now consider the project requirements in terms of
Modula-2's separate compilation facilities. Subsystems
will most likely be composed of one or more compila-
tion units. Defining and maintaining consistent inter-
faces is of critical importance in ensuring error-free

communication between subsystems. During the de-
sign stage, however, the subsystems themselves do not
yet exist. They are known only by their interfaces.

The concept of a subsystem interface corresponds
to the definition module construct. Thus, interfaces can
be defined as a set of definition modules before sub-
system development (i.e., design and coding of the
implementation modules) begins. These modules are
distributed to all members of the programming group,
and it is through these modules that inter-subsystem is
defined. Interface consistency is automatically en-
forced by the compiler.

Modula-2 Libraries

The library is a collection of separately compiled
modules that forms an essential part of most Modula-2
implementations. It typically contains the following
types of modules:

1. Low-level system modules which provide ac-
access to local system resources;
2. Standard utility modules which provide a con-
sistent system environment across all Modula-2
implementations;
3. General-purpose modules which provide useful
operations to many programs;
4. Special-purpose modules which form part of a
single program;

The library is stored on one or more disk files con-
taining compiled forms of the library module's compi-
lation units. The library is accessed by both the compiler
and the program loader — the former reads the com-
piled definition modules while compiling and the latter
loads the compiled implementation modules when ex-
ecuting the program that imports library modules.

A dependency arises between library modules and
the modules that import them. Consider the example of
a single library module. The compiler must reference
the module's symbol file (a compiled definition module)
in order to compile the implementation module. There-
fore, the definition module must be compiled first. Once
an implementation module has been compiled, its ob-
ject file is tied to the current symbol file, as the object
code is based on procedure and data offsets obtained
from the symbol file. Similarly, when a program im-
ports a library module, it is assumed that the symbol
file offsets are accurate reflections of the corresponding
object file.

The Modula-2 language contains no standard pro-
cedures for I/O, memory allocation, or process sched-
uling. Instead, these facilities are provided by standard
utility modules stored in the library. Standard utility
modules are expected to be available in every Modula-
2 implementation. Thus, by using only standard mod-
ules, Modula-2 programs become portable across all
implementations.

The advantages of expressing commonly-used
routines as library modules (rather than part of the lan-
guage) include a smaller compiler, smaller run-time
system, and the ability to define alternative facilities
when the standard facilities prove insufficient. Disad-
vantages include the need to explicitly import and bind
library modules, and occasionally a less flexible syntax imposed by expressing standard routines as library modules (as opposed to their being handled specially by the compiler).

CONCLUSION

The examples cited above can only provide a clue as to the power and flexibility of the Modula-2 language. It is the hope of the authors that they can pique significant curiosity and interest into this amazing new programming tool.

REFERENCES

1. Niklaus Wirth, Programming in Modula-2, Springer-Verlag, 1982
This book is intended as a text for a first course in data structures that is also a second course in programming. It presents all of the major data structures including stacks, queues, lists, trees, and graphs and describes recursion, list processing, sorting, and searching. An appendix provides a brief tutorial on Pascal. The emphasis is on practical techniques as opposed to theoretical concepts. All algorithms and examples are presented in Pascal.

This book is excellent both for students and for practicing programmers who want to learn how to apply algorithms and data structures, whether or not they use Pascal. However it would not be appropriate, nor was it intended to be, for those merely wishing to learn Pascal.

The authors employ several pedagogical techniques which others would do well to emulate. First numerous examples and sample programs are presented; the authors do not merely rely on textual explanations. In spite of this, there are very few typographical or algorithmic errors as so often is the case with multiple figures. Second, the same basic figure is repeated several times with each version successively updated to show the intermediate results of an algorithm. For example, an array is listed after each pass of a sorting algorithm so the reader can follow how the sort progresses. Third, algorithms are often presented as a combination of Pascal and pseudocode, thus highlighting the key points and not confusing the reader with such irrelevancies as initialization or I/O. Fourth, algorithms are presented several times with each new version a refinement of the previous one.

My only criticisms would be that some of the algorithms could be simplified, frequently by more appropriate tests in “while” statements; and more use should be made of enumeration types — certain algorithms had a Fortran ring to them. However these are nitpicks: the book is excellent and is highly recommended to all PUG’ers.

Arthur Salwin
1405 Homeric Ct.
McLean, VA 22101
SOFTWARE BUILDING BLOCKS, INC.
ANNOUNCES PASCAL COMPILER FOR THE
IBM PERSONAL COMPUTER®

ITHACA, NY — A new company, Software
Building Blocks, Incorporated, has been formed in Ithaca,
New York. The founders of the company are Jeff
Moskow, author of the popular, highly acclaimed Pas-
call/Z® compiler marketed by Ithaca InterSystems, Inc.;
Laurie Hanselman Moskow, formerly Software Prod-
ucts Manager at InterSystems; and William Kellner, a
software engineer who has worked extensively with
Moskow on the Pascal/Z compiler.

The first product to be released by Software Build-
ing Blocks, Inc. is a two-pass, locally optimizing
compiler for the IBM Personal Computer. The initial
release will run under PC-DOS®; and a CP/M-86® ver-
sion is planned for the near future. Based on the Pas-
call/Z compiler, the Software Building Blocks
implementation, SBB Pascal®, closely follows the Jensen & Wirth
definition of the language, with extensions designed to
aid the professional programmer in serious software
development. Extensions will include: variable length
strings, direct file access, arbitrary precision BCD
numbers for business arithmetic, functions returning
structured values, separate compilation, externen-
strings, direct file access, arbitrary precision BCD
numbers for business arithmetic, functions returning
structured values, separate compilation, external
characters, finding and/or replacing of strings, copying lines
of text, autoindent for entering structured programs, and
many other features. The editor makes use ofthe
SAGE OPENS BOSTON DIVISION

Sage Computer Technology, headquartered in
Reno, Nevada, has announced the opening of its Bos-
ton division.

The purpose of the new facility is to provide re-
regional support for dealers and users of the Sage line of
16-bit microcomputers, and to expedite delivery of new
units throughout the Eastern United States.

A complete inventory of Sage II's, Sage IV's, parts
and literature is stocked, and a fully-equipped and
staffed service department is maintained on the
premises.

According to Rod Coleman, Sage president, plans
call for a total of nine such offices to augment the com-
pany's domestic sales and support activities. "Regional
support for our dealers and OEMs is a critical part of
our marketing plan."

Sage's Boston office is now open to dealers &
OEMS, and is located at 15 New England Executive
Park, Suite 120, Burlington, MA 01803. The telephone
number is (617) 229-6868.

More information about Sage micros is available
from either Boston office or corporate office at 4905
Energy Way, Reno, Nevada 89502. Telephone (702) 322-
6868.

If agency contact is required, phone or write The Schraff
Group, 18226 W. McDurmott, Suite E, Irvine, CA
92714. Telephone (714) 540-8977.

NEW, 16-BIT SAGE IV
HAS WINCHESTER PLUS
MULTI-USER CAPABILITY

RENO, NEVADA — Sage Computer Technology
has announced availability of the Sage IV, 16-bit (68000)
supermicro.

The new multi-user computer, which accommod-
dates up to 6 simultaneous users, surpasses the consid-
erable capabilities of the Sage II introduced in March,
1982.

Both machines are based on the 8 MHz 68000 pro-
cessor, and both are capable of performing 2-million
operations per second. According to Rod Coleman, Sage
president, they offer performance comparable to that
of high-end minicomputers at a mid-range to high-end
business micro price.

The Sage IV comes standard with 128K of main

© IBM and IBM Personal Computer are registered
trademarks of International Business Machines
© Pascal/Z is a trademark of Ithaca InterSystems, Inc.
© PC-DOS is a trademark of International Business
Machines Corporation
© Software Building Blocks and SBB Pascal are trade-
marks of Software Building Blocks, Inc.

® CP/M-86 is a trademark of Digital Research, Inc.
memory which is expandable, optionally to a megabyte. This represents an enormous jump from the 128K to 512K expandability of the Sage II, which in turn offers far greater capacity than the typical 64K, 8-bit computer.

In addition, a 5 to 30Mb Winchester disk, either fixed or removable, is built into the Sage IV next to a 5¼ inch floppy backup. Since there are no wait states, a 20K program loads from the floppy in 1 second, and from the hard disk in 1/10 second.

The cabinet, though about 1½ inches taller than that of the Sage II, is still deceptively small, measuring only 6½" high, 12½" wide and 16¾" deep.

"There aren't any tradeoffs with either of these machines, said Coleman, "the user doesn't have to give up software support to get high performance, because the Sages' p-System standard operating system is able to run hundreds of popular programs developed for 8-bit micros."

More information may be had by contacting Sage Computer Technology, 35 North Edison Way, Suite 4, Reno, Nevada 89502. Telephone (702) 322-6868.

If agency contact is required, phone or write The Schraff Group, 1325 Airmotive Way, Suite 175, Reno, Nevada 89502. Telephone (702) 348-7339.

---

NEW MODULA-2 MANUAL FEATURES

TUTORIALS, STANDARD LIBRARY

DEL MAR, CA, Jan. 21 — A 264-page Modula-2 user's manual, featuring a language tutorial and standard library definitions, is now available from Volition Systems here.

Modula-2 is a new programming language designed by Niklaus Wirth to replace his earlier language, Pascal, in a wide range of real-world applications. Together with Wirth's own specifications of the language, this manual provides a complete description of Volition's implementation of Modula-2, according to its author Richard Gleaves of Volition Systems.

The manual is designed to be used with Wirth's 48-page monograph which defines Modula-2 in a concise but informal style. The monograph is included with the manual. Wirth's newly published book Programming in Modula-2 is also available from Volition.

The manual contains a tutorial for Pascal programmers that can make them comfortable with the language within a few hours and proficient within a week, Gleaves said.

The name Modula-2 comes from MODUlar LAn­
guage. It uses modules to facilitate the development and maintenance of large, complex systems. The language is especially useful in large industrial and commercial applications where it can save software developers both time and money.

Modula-2 is designed to utilize standard software modules, which are defined in the new manual. These modules provide access to the facilities normally pro-

vided by an operating system, such as program and pro-

cess control; console and file I/O, including random

access files and disk directory operations; and storage

management. The standard software modules also in-
clude utility routines for format conversion, strings, 19
digit BCD arithmetic, and other facilities.

The manual is divided into six sections. The Modu-
la-2 tutorial for Pascal programmers comprises about
one-third of the book.

In addition, there is an introductory section and
sections defining the standard library modules, the util-
ity library, a system document that describes the im-
plementation of Modula-2 for UCSD PascaII and a
machine-specific implementation guide which includes
information on machine specific library modules, in-
trrupt handling, and machine-level data representation.

The Modula-2 User's Manual, including Wirth's
Modula-2 report, is immediately available from Voli-
ton Systems, P.O. Box 1236, Del Mar, CA 92014 for
$35 per copy. Wirth's book, Programming in Modula-
2, published in 1982 by Springer-Verlag, can be ordered
for $16. Further information about the programming
language is also available from Volition Systems.

Volition Systems concentrates on systems soft-
ware development and on research and development in
hardware and software. Since the company was founded
in 1980, it has been a leader in the implementation and
dissemination of the Modula-2 language and other high
level languages and in the design and development of
advanced computer architectures.

For further information, contact:
Volition Systems
P.O. Box 1236, Del Mar, CA 92014
(619) 481-2286
UCSD Pascal is a trademark of the Regents of the University of California.

MODULA-2 USER'S MANUAL from Volition Systems (Del Mar, CA) describes Niklaus Wirth's new programming language in a 264-page loose-leaf format. This document contains a complete tutorial for Pascal programmers, sections defining the standard library modules and the utility library, and an implementation guide. The manual comes with a copy of Wirth's 48-page technical report on Modula-2.

Modula-2 is particularly useful in large industrial and commercial applications where using standard modules facilitates development of large, complex sys-

tems, according to Volition, which has pioneered in
commercial implementations of the new language. The
Modula-2 User's Manual is immediately available from
Volition Systems, P.O. Box 1236, Del Mar, CA 92014
for $35.

For further information, contact:
A. Winsor Brown
(714) 891-6043
WASHINGTON, D.C., June 3 — USUS, Inc., the UCSD Pascal User’s Society, will hold its semi-annual national meeting at the Crystal City Hyatt Hotel here October 14-16, according to Robert Peterson, USUS president.

In conjunction with the meeting, USUS will sponsor two free tutorials — an introduction to the p-System and an introduction to UCSD Pascal, including Apple Pascal.*

The meeting will feature technical presentations, hardware and software demonstrations, language tutorials, special interest group meetings and software library exchange. Also planned are expert user and major vendor panels. Election of officers will be held.

“Non-USUS members are welcome to register and attend any or all of the meeting programs,” Peterson noted.

USUS (pronounced use-us) represents users of the UCSD Pascal System and its derivatives including the UCSD p-System and Apple Pascal. It is the most widely-used, machine-independent software system. The society is non-profit and vendor independent.

The UCSD Pascal System has more than 100,000 users and is capable of running on nearly any computer. It was developed at the University of California San Diego to facilitate software portability.

Among the special interest group meetings scheduled for the Washington meeting are those for users of IBM Personal Computers, Apple, DEC, Texas Instruments, NEC Advanced Personal Computer, the IBM display writer and Sage Computer Technology computers.

Also meeting will be those interested in application development, graphics, communications, file access, Modula-2, UCSD Pascal compatibility and the Advanced System Editor.

The software library, with significant recent acquisitions, will be available for reproduction on various diskette formats. Members at the meeting will be able to copy the library onto their own disks for $1.00 each.

Those registering for the meeting before September 23 will qualify for the pre-registration price of $25. Checks should be made payable to USUS and mailed to USUS Meeting Committee, P.O. Box 1148, La Jolla, CA 92038. Registration at the door will be $35 and will begin at 10 a.m. Friday, October 14.

Hotel reservations should be made directly with the Crystal City Hyatt hotel (adjacent to Washington National Airport), 2799 Jefferson Davis Highway, Arlington, VA 22202, (703) 486-1234. Additional meeting information is available from Thomas Woteki, Information Systems Inc., 3865 Wilson Blvd., Suite 202, Arlington, VA 22203, (703) 522-8898.

USUS was created to promote and influence the development of the UCSD Pascal System and to provide users and vendors with a forum for education and information exchange about it. Annual membership in the society is $25 for individuals and $500 for institutions.

* Apple Pascal is a trademark of Apple Computer, Inc.

SAN FRANCISCO, CA, June 15, 1983 — A special interest group (SIG) for users of the Advanced System Editor (ASE) for the UCSD Pascal System has been formed by USUS, the UCSD Pascal System User’s Society, according to Robert W. Peterson, president of the society.

The new SIG will be chaired by Sam Bassett, of San Francisco, CA. “The ASE SIG will be open to any USUS member who is using or thinking about getting the Advanced System Editor.” Peterson said.

The new ASE SIG allows members to share common problems and solutions and will serve as a clearing house for information relating to implementation, optimization and use of ASE on a variety of systems which have the UCSD p-System installed.

The SIG has established a liaison with Volition Systems of Del Mar, CA, the creators of ASE. It will coordinate relevant contributions to the USUS Software Exchange Library and to USUS News, the society’s quarterly newsletter, Bassett said. Furthermore, SIG members may communicate via electronic mail under USUS sponsorship.

The next ASE SIG meeting will take place at the USUS semi-annual national meeting in Washington, D.C., October 14-16. In addition to the ASE and other SIG sessions, the USUS meeting will feature tutorials on UCSD Pascal and the UCSD p-System. Also on the agenda are technical presentations, software exchange, hardware and software demonstrations and an expert user panel.

Membership in the ASE SIG is free of charge to any member of USUS, the vendor-independent, nonprofit user’s group for the UCSD Pascal System. Annual membership in the society is $25 for individuals and $500 for institutions.

USUS (pronounced use-us) was founded in 1980 to promote and influence the development of the UCSD Pascal System and to provide a forum for education and information exchange about it. Further information on USUS is available from the Secretary, USUS, P.O. Box 1148, La Jolla, CA 92038.

SAN DIEGO, CA, May 26 — USUS, the UCSD Pascal System User’s Society, has formed a special interest group (SIG) for users of the new Modula-2 programming language, according to Robert W. Peterson, USUS president.

The new SIG will be chaired by David Ramsey of Corvus Systems, Inc. (San Jose, CA). The group was formed when USUS held its semi-annual national meeting here last month. Modula-2 runs on Version II based UCSD Pascal Systems.

“The Modula-2 SIG will be open to any USUS member using or wanting to investigate this language,” Ramsey said. “It is, to my knowledge, the first user’s group devoted to communication about Modula-2.”
The new language was created by Niklaus Wirth to answer difficulties encountered with his earlier language, Pascal. "As people discover the benefits of working in this new language, we expect this SIG to expand rapidly," Ramsey said.

Implementations of the Modula-2 programming language are available for the Apple II, //e and /// computers, the IBM Personal Computer, the 68000-based Sage 2 and 4, the Texas Instruments 9900, the Scenic One and Z80/8080-based systems, according to Joel J. McCormack of Volition Systems (Del Mar, CA).

Volition is the only current supplier of the language for use on microcomputers and supplies systems as well as the Modula-2 language to run on them.

"Because the language is modular, users spend less time writing and maintaining code," McCormack said. "Standard library modules provide Modula-2 with a standard operating environment, and programs created within it are portable across all Modula-2 systems."

The new Modula-2 SIG will enable users to share experiences with others using the language or developing applications in it, Ramsey said. "We expect to serve as a clearing house for user information in this fast-changing area."

One of the first goals of the SIG is creation of a user’s library of Modula-2 programs that will be included in the USUS library, Ramsey noted. It will be compiled by Curt Snyder of Allergan Pharmaceuticals (Irvine, CA).

Membership in the Modula-2 SIG is free of charge to any member of USUS, which is the vendor-independent, non-profit user’s group for the UCSD Pascal System. Annual membership in the society is $25 for individuals and $500 for institutions. Further information on USUS is available from the Secretary, USUS, P.O. Box 1148, La Jolla, CA 92038.

For those wanting to know more about the Modula-2 SIG, Ramsey can be reached at Corvus Systems, 2029 O’Toole Avenue, San Jose, CA 95131, (408) 946-7700, extension 267.

Volition Systems has pioneered in the implementation and dissemination of the Modula-2 language. Further information about Modula-2 and available implementations may be obtained from Tracy Barrett, Volition Systems, P.O. Box 1236, Del Mar, CA 92014, (619) 481-2286.

---

**PASCAL USERS, VENDORS GATHER FOR USUS SAN DIEGO MEETING**

SAN DIEGO, CA, May 2 — USUS, the UCSD Pascal System User’s Society, formed five new special interest groups (SIG’s) and made plans for a first regional chapter at its well-attended, semi-annual national meeting here last week, according to Robert W. Peterson, USUS president.

In addition, two vendors of UCSD Pascal products — Apple Computer, Inc. and Volition Systems — chose the occasion to reveal new offerings.

"Record meeting attendance reflects the users’ commitment to increased knowledge about use of the UCSD Pascal System," Peterson said. "More than 240 attended and actively participated in special interest group and committee meetings, panel discussions and the four tutorials."

Keynote speaker for the event was Andrew Greenberg, designer and co-author of the popular Wizardry games. He told how he had solved the challenge of putting a very large program like Wizardry on a microcomputer with limited disk and main memory storage.

"Greenberg offered members valuable insights into program design, structure and implementation," Peterson noted.

The new special interest groups are for application developers and for users of the NEC Advanced Personal Computer, the IBM Display Writer, the Advanced System Editor from Volition Systems, and the Modula-2 programming language. In addition, plans for the national organization’s first local group in Southern California were discussed.

USUS already has SIG’s for users of Apple, DEC, Texas Instruments and Sage computers, the IBM Personal Computer and for those interested in communications, word processing and UCSD Pascal compatibility.

Of particular interest to those attending the meeting was the demonstration area, where the latest advances in UCSD Pascal hardware, software and applications were demonstrated on 20 different machines, Peterson said.

At the meeting, Apple Computer, Inc., which has an installed base of some 82,000 Pascal development systems on its Apple II and Apple /// computers, announced that updates of Apple II Pascal and Apple /// Pascal will be available this year.

Apple revealed that Version 1.2 of Apple II Pascal will be available in the fourth quarter of 1983 and will provide support for all features of the Apple Ile including extended memory support for the 128K Ile. Version 1.2 also makes available facilities for integrating into the UCSD Pascal environment in a natural way additional mass storage devices such as hard disks.

Apple also confirmed that Version 1.1 of Apple /// Pascal will be available at the end of June 1983. Its most notable feature is the Standard Apple Numeric Environment that fully implements the IEEE standard for floating point arithmetic.

Volition Systems demonstrated the new Modula-2 programming language running for the first time on an IBM Personal Computer. USUS members formed a Modula-2 SIG at the meeting to exchange information about the language. It will be chaired by Dave Ramsey, Volition Systems (San Jose, CA).

The chairman of the newly formed application developer’s SIG is Dennis Gallinat, Apple Computer (Cupertino, CA), and Samuel Bassett, Bassett Information Processing (San Francisco, CA) is chairing the Advanced System Editor SIG.

Lane Sharman, Resource Systems Group (Del Mar, CA) will head the Special Interest Group for the IBM Display Writer, and the NEC Advanced Personal Computer SIG will be chaired by George Symons, TICOM Systems, Inc. (Marina del Rey, CA).

The fall USUS meeting will be held in Washington, D.C., at the Hyatt Regency Crystal City, October 14-16, 1983. Further information is available from the Sec-
USUS, P.O. Box 1148, La Jolla, CA 92038.

USUS (pronounced use-us) is a vendor-independent, non-profit user's group for the most widely used, machine-independent software system — the UCSD Pascal System, and its successors such as the Apple Pascal System and the UCSD p-System.

USUS was created to promote and influence the development of the UCSD Pascal System and to provide a forum for education and information exchange about it. USUS has institutional as well as individual membership in more than 20 countries. Annual membership in the society is $25 for individuals and $500 for institutions.

VOLITION DEMONSTRATES
MODULA-2 FOR IBM PC

DEL MAR, CA, May 3 — Volition Systems here has demonstrated Niklaus Wirth's new Modula-2 programming language running for the first time on the IBM Personal Computer.

The new implementation was demonstrated for members of USUS, the UCSD Pascal System User's Society, at its semi-annual national meeting in San Diego last week. Modula-2 will be included as part of Volition's complete software development system.

"Modula-2 is proving especially valuable in large industrial and commercial applications where standard software modules can save time and money in program development and maintenance," according to Joel J. McCormack of Volition Systems.

"Now our new implementation will make these savings possible on the IBM PC. Our software development system will even run efficiently on 64K PC's," he continued. "And the availability of Modula-2 on the IBM PC should make the language even more attractive to application developers."

The IBM PC implementation will significantly expand the availability of Modula-2. Current Volition versions are based on the 6502 (including Apple II2 and Apple /// computers), the 8080/Z80, TI 9900, and the 68000.

Niklaus Wirth developed Modula-2 (from MODular LAnguage) to replace his earlier language, Pascal. Whereas Pascal was intended as a teaching language, Modula-2 is expressly designed for use in a wide range of real-world applications, and it offers great flexibility in the development of large, complex systems.

The implementation for the IBM PC is expected to be available in the third quarter of 1983, McCormack said. The system will include Modula-2 and Pascal compilers, the modula library, the powerful ASE text editor, V-NIX® command shell (that provides a UNIX3-like programming environment), and a complete set of utility programs for file manipulation and electronic mail communication.

Volition Systems concentrates on systems software development and on research and development in hardware and software. Since the company was founded in 1980, it has led in the implementation and dissemination of the Modula-2 language and other high-level languages and in the design and development of advanced computer architectures.

For further information, contact:
Volition Systems
P.O. Box 1236, Del Mar, CA 92014
(619) 481-2286

1 UCSD Pascal is a trademark of the Regents of the University of California.
2 Apple II and Apple /// are trademarks of Apple Computer, Inc.
3 UNIX is a trademark of Bell Laboratories.
0. **DATE**  Apr. 28, 1983

1. **IMPLEMENTOR/MAINTAINER/DISTRIBUTOR** (*Give a person, address and phone number.*)

   Robert Reimiller
   OmegaSoft
   P.O. Box 842
   Camarillo, CA 93010
   (805) 987-6426

2. **MACHINE/SYSTEM CONFIGURATION** (*Any known limits on the configuration or support software required, e.g. operating system.*)

   Running Moos, OS-9, or Flex OS
   Requires 48K to 56K (Recommended)

3. **DISTRIBUTION** (*Who to ask, how it comes, in what options, and at what price.*)

   North America: From Omega Soft
   International: From OmegaSoft or distributors in Germany, Switzerland, Great Britain, Australia, Sweden, and the Netherlands. Price is $425 to $475 for Compiler, Debugger, and Runtime.
   Relocatable Assembler/Linker available for $125 to $150.

4. **DOCUMENTATION** (*What is available and where.*)

   220 pg. Pascal manual with complete syntax and installation instructions.

5. **MAINTENANCE** (*Is it unmaintained, fully maintained, etc?*)

   Yearly maintenance is $100 to $125

6. **STANDARD** (*How does it measure up to standard Pascal? Is it a subset? Extended? How.*)

   Complete ISO standard except packed variables and procedural parameters. Scored 92% on conformance section of validation suite.
   ISO report in manual. Extended for real time and industrial control applications.

7. **MEASUREMENTS** (*Of its speed or space.*)

   Warshall's Algorithm: procedure size=270 bytes,
   Execution time=9.7 seconds

8. **RELIABILITY** (*Any information about field use or sites installed.*)

   Over 400 sites installed.
   Over 4000 sites installed.

9. **DEVELOPMENT METHOD** (*How was it developed and what was it written in?*)

   From scratch in assemble language.

10. **LIBRARY SUPPORT** (*Any other support for compiler in the form of linkages to other languages, source libraries, etc.*)

    Optional libraries to handle AMD9511 APU CHIP, and Multi-Tasking Primitives.
OmegaSoft Pascal Version 2

Pascal Processor Identification

Host Computer: Smoke Signal Broadcasting Chief-tain 9522812W10 running the OS-9 operating system.
Host Computer Requirements: MC6809 processor, minimum of 48K bytes of memory, 2 or more disk drives, running the OS-9, MDOS, XDOS, DOS69, or FLEX operating system.
Processor: OmegaSoft pascal version 2.10

Test Conditions

Tester: R. D. Reimiller
Date: June 1982
Validation Suite Version: 3.0

General Introduction to the OmegaSoft Implementation

The OmegaSoft Pascal compiler was developed to provide the users of the 6809 processor with a fast and efficient way to develop code capable of running on the host development system or installed into a target system. The compiler is aimed primarily at industrial applications such as process control and instrumentation. Due to the nature of these applications many extensions were added such as byte arithmetic, long integers, dynamic length strings, modular compilation, and versatile variable addressing. As a secondary requirement it was desired that the compiler be able to accept a Pascal program written in ISO standard Pascal wherever possible.

CONFORMANCE TESTS

Number of tests passed = 144
Number of tests failed = 12 (9 reasons)

Details of Failed Tests

Test 6.4.2.3-3: If an enumerated type is defined in the index declaration part of an array its values cannot be referenced until the array declaration is complete.
Test 6.4.2.3-4: If an enumerated type is defined in a record its values cannot be referenced until the record declaration is complete.
Tests 6.6.3.1-4, 6.6.3.4-1, 6.6.3.4-2, and 6.6.3.5-1: Procedures and functions cannot be passed as parameters.
Test 6.6.5.4-1: Pack and Unpack procedures are not supported.
Test 6.7.2.2-3: Failed on MOD using a negative dividend. The Jensen/Wirth “remainder after division” method is used rather than the method specified in the ISO standard.
Test 6.8.2.4-1: Non-local GOTO’s are not allowed.
Test 6.8.3.9-1: Assignment to the control variable of a FOR loop occurs after the evaluation of the first expression.
Test 6.9.3-1: Standard I/O devices may not be re-defined if declared.
Test 6.9.3.5.1-1: Real numbers written out in floating point format always have six digits to the right of the decimal point.

DEVIANCE TESTS

Number of deviations correctly detected = 83
Number of tests showing true extensions = 45 (22 reasons)
Number of tests not detecting erroneous deviations = 9 (6 reasons)

Details of Extensions

Test 6.1.5-4: No digits are needed after the decimal point in a real number.
Tests 6.1.6-4 and 6.1.6-5: Labels may be a positive integer constant.
Tests 6.1.7-5, 6.4.3.1-3, 6.4.3.1-4, 6.6.3.3-5, 6.9.3.2-2: All variables are packed at the byte level, the reserved word “Packed” is ignored in any type declaration.
Tests 6.1.7-6, 6.1.7-7, 6.1.7-8, 6.4.3.2-5: Strings, characters, and arrays of less than 127 elements are all compatible.
Tests 6.1.7-11 and 6.4.5-12: Strings are dynamic length, allowable length is from 0 (null string) to 126.
Tests 6.2.1-8 and 6.2.1-10: Label, const, type, and var declaration sections can be in any order and repeated multiple times until a procedure/function declaration or “begin” is encountered.
Test 6.3-9: In any context where a constant is acceptable an expression with a constant value may be used.
Test 6.4.2.3-5: All enumerated type values are compatible.
Test 6.4.3.3-8: The values of the case constants in a record variant declaration are not used, access is provided to all variants at all times.
Tests 6.4.5-7: All subranges of the same type are compatible.
Tests 6.4.5-8 and 6.4.5-13: Arrays of the same size are compatible.
Tests 6.4.5-9 and 6.4.6-7: Records of the same size are compatible.
Test 6.4.5-10: All pointers are compatible with other pointers or the type “Hex”.
Test 6.6.2-5: Any type with a size of less than 128 bytes can be used as a function return type.
Test 6.6.6.3-2: Trunc and round can have integer or long integer parameters.
Test 6.7.2.3-2: Logical operators are valid for character and integer expressions.
Test 6.7.2.5-6: Arrays of the same size can be compared. Records of the same size can be compared.
Test 6.8.2.4-2: Goto between branches of an If statement are allowed.
Test 6.8.2.4-3: Goto between branches of a Case statement are allowed.
Tests 6.8.3.5-7 and 6.8.3.5-8: Subrange Case statement constants are allowed.
Tests 6.8.3.9-5, 6.8.3.9-6, 6.8.3.9-7, 6.8.3.9-10, 6.8.3.9-12, 6.8.3.9-13, 6.8.3.9-14, 6.8.3.9-15, 6.8.3.9-16, and 6.8.3.9-17: No restrictions are placed on For statement control variable.
Tests 6.8.3.9-8 and 6.8.3.9-9: If a For statement is entered and exited normally the control variable will be valid and contain the final value. If a For statement is not entered then the control variable will be valid and contain the initial value.

Details of Deviations

Test 6.1.8-5: A number can be terminated by a letter.
Tests 6.2.1-5 and 6.2.1-6: Multiple siting for labels is not checked, nor are labels required to be sited at all.
Tests 6.2.2-8, 6.3-6, and 6.4.1-3: Error in scope rules.
Test 6.6.1-7: Unresolved forward function or procedure declaration is not detected.
Test 6.6.3.3-4: Use of a field selector as a parameter is not detected.
Test 6.10-4: No check is made for duplication of program parameters.

ERROR-HANDLING

Number of errors correctly detected = 19
Number of errors not detected = 31 (13 reasons)

Details of Errors Not Detected

Tests 6.2.1-11, 6.4.3.3-11, 6.4.3.3-12, 6.4.3.3-11, 6.5.4-2, and 6.6.2-9: No checking is made to verify whether or not a variable is accessed that has an undefined value. Instead the variables are guaranteed to contain garbage unless initialized.
Tests 6.4.3.3-1, 6.6.5.3-8, 6.6.5.3-9, and 6.6.5.3-10: Any tagfields or selector variables in a record variant are irrelevant to which variants can be accessed.
Test 6.4.6-10: No subrange checking on parameter passing.
Tests 6.4.6-12, 6.4.6-13, and 6.7.2.4-4: Overflow checking is done on sets based on byte count — not per element.
Tests 6.5.4-1, 6.6.5.3-4, 6.6.5.3-5, and 6.6.5.3-11: Pointer value is not checked before use.
Tests 6.5.5-2, 6.5.5-3, 6.6.5.3-6, and 6.6.5.3-7: There are no restrictions on the use of pointers or file buffer variables which are currently parameters or elements of a with statement.
Test 6.6.5.2-5: To support random files a “get” is not executed until called as a procedure or when accessing the file buffer without a valid element — not at the time of “reset”.
Test 6.6.6.4-7: Char and Hex variables “roll over” from maximum value to zero — it is not considered an error.
Test 6.6.6.5-7: If eof is true — so is eoln — it is not considered an error to check eoln if eof is true.
Tests 6.8.3.5-10 and 6.8.3.5-11: If no match in case statement, falls through with no error.
Test 6.8.3.9-18: No restrictions on the control variable of a For loop.
Test 6.8.3.9-1: At the completion of a For loop the control variable is valid and has the final value.
Tests 6.9.3.2-5 and 6.9.3.2-5: Writing of real numbers with no digits past the decimal point is permissible.

QUALITY MEASUREMENT

Number of tests run = 52
Number of tests incorrectly handled = 5

Results of Tests

“Synthetic Benchmark” — execution time 1 minute, 10 seconds.
“GAMM measure” — execution time 1 minute, 40 seconds for N = 1000
procedure calls — execution time 40 seconds
identifiers are significant up to 120 characters.
source lines may be up to 120 characters.
no reasonable limit on number of real literals allowed.
no reasonable limit on number of strings allowed.
if a line of code is incorrectly part of an unclosed comment the compiler will signal that no code was generated for the line.

at least 50 types may be declared in a program.
no reasonable limit on number of labels, but there can be a maximum of 8 forward referenced goto’s in a block.

at least 128 constant definitions are allowed per constant declaration part.
at least 128 procedures are permitted in a program.
maximum size for an array or record or for any variable section is 32750 bytes.
at least 8 index types can appear in an array type.
at least 128 case-constant values are permitted in a variant record.
at least 50 record-sections can appear in the fixed part of a record.
at least 30 distinct variants are permitted in a record.
“Warshall’s algorithm” procedure size = 270 bytes, execution time = 9.7 seconds.
considerably less than 300 identifiers are allowed in a declaration list (actual number depends on length of identifier).
at least 8 dimensional array is allowed.
procedures may be nested to at least 15 levels.
at least 30 formal parameter sections can appear in one parameter list.
the dispose in the standard heap manager is a dummy, a more complex heap manager is available.
deeply nested function calls are allowed (at least 6).
deeply nested compound statements are allowed (at least 25).
deeply nested procedure may have at least 300 statements.
deeply nested if statements are allowed (at least 25).
at least 256 case constants are allowed.
at least 300 constants are allowed in a case-constant list.

case statements can be nested to at least 15 deep.
repeat loops can be nested to at least 15 deep.
while loops can be nested to at least 15 deep.
for loops can be nested to at least 15 deep.
with statements can be nested to at least 15 deep.
recursive I/O can be used with the same file for the second I/O action.
at least 30 variable-accesses can appear in a read or readln parameter list.
at least 30 write-parameters can appear in a write or writeln parameter list.
data written on the output field appears regardless of the omission of a line marker.

IMPLEMENTATION-DEFINED
Number of tests run = 12
Number of tests incorrectly handled = 1

Details of Implementation-Defined Features

Tests 6.1.9-5 and 6.1.9-6: alternate symbols are available for comments, array indices, and pointers.
Test 6.4.2.2-10: Maxint is 32767
Test 6.4.3.4-5: maximum range of set elements is 0..1007
Test 6.6.6.2-11: Base = 2, Bits of mantissa = 24, not rounding, minimum value = 2.710506E-20, maximum value = 9.223372E+18
Tests 6.7.2.3-3 and 6.7.2.3-4: Boolean expressions are fully evaluated.

Tests 6.8.2.2-1 and 6.8.2.2-2: In an assignment statement evaluation of the expression is done before the selection of the variable.
Test 6.8.2.3-2: When a procedure is called the parameters are evaluated in forward order.
Test 6.9.3.2-6: Default field widths are: Integers = 10, Boolean = 6, Real = 16, Longinteger = 16, Hex = 6.
Test 6.9.3.5.1-2: Real values written in floating point format have 2 exponent digits.
Test 6.9.3.6-1: Boolean values written in the default fieldwidth have the format as shown (between quotes) "TRUE" and "FALSE".

Details of Tests Incorrectly Handled

Tests 6.6.6.1-1: Functions are not allowed to be passed as parameters to a procedure.

Level 1 Tests — Not applicable

EXTENSIONS

Extension present = 1

Result of Extension

Test 6.8.3.5-16: An otherwise clause is allowed on a case statement.
Please enter my membership in Pascal Users Group. I understand I will receive “Pascal News” whenever it is published in this calendar year.

Pascal News should be mailed
1 yr. □ in USA $25 □ outside USA $35 □ AirMail anywhere $60
3 yr. □ in USA $50 □ outside USA $80 □ AirMail anywhere $125

(Make checks payable to: “Pascal Users Group,” drawn on USA bank in US dollars)

Enclosed please find US $_____.
on check number ______

(I have difficulty reading addresses. Please forgive me and type or print clearly.)

My address is:

NAME

ADDRESS

PHONE

COMPUTER

DATE

□ This is an address correction here is my old address label:

________________________________________________________________________

________________________________________________________________________

________________________________________________________________________

Coupons
JOINING PASCAL USER GROUP?

• Membership is open to anyone: Particularly the Pascal user, teacher, maintainer, implementor, distributor, or just plain fan.
• Please enclose the proper prepayment (check payable to “Pascal User’s Group”).
• When you join PUG any time within a year: January 1 to December 31, you will receive all issues Pascal News for that year.
• We produce Pascal News as a means toward the end of promoting Pascal and communicating news of events surrounding Pascal to persons interested in Pascal. We are simply interested in the news ourselves and prefer to share it through Pascal News. We desire to minimize paperwork, because we have other work to do.

RENEWING?

• Please renew early (before November) and please write us a line or two to tell us what you are doing with Pascal, and tell us what you think of PUG and Pascal News.

ORDERING BACK ISSUES OR EXTRA ISSUES?

• Our unusual policy of automatically sending all issues of Pascal News to anyone who joins within a year means that we eliminate many requests for backissues ahead of time, and we don’t have to reprint important information in every issue — especially about Pascal implementations!
• Issues 1 . . 8 (January, 1974 — May 1977) are out of print.
• Issues 9 . . 12, 13 . . 16, & 17 . . 20, 21 . . 23 are available from PUG(USA) all for $25.00 a set.
• Extra single copies of new issues (current academic year) are: $10 each — PUG(USA).

SENDING MATERIAL FOR PUBLICATION?

• Your experiences with Pascal (teaching and otherwise), ideas, letters, opinions, notices, news, articles, conference announcements, reports, implementation information, applications, etc. are welcome. Please send material single-spaced and in camera-ready (use a dark ribbon and lines 15.5 cm. wide) form.
• All letters will be printed unless they contain a request to the contrary.
Back issues are requested and sent in sets

- $15 □ set 0 Issues 1 . . . 8 (January 1974 — May 1977) OUT OF PRINT
- $25 □ set 2 Issues 13 . . . 16 (December 1978 — October 1979)

Requests from outside USA please add $5 per set.

All memberships entered in 1983 will receive issue 24 and all other issues published in that year.
Make check payable to: “Pascal Users Group,” drawn on USA bank in US dollars.

Enclosed please find US $____ . ____
on check number ______

(I have difficulty reading addresses. Please forgive me and type or print clearly)

My address is:

NAME __________________________________________
ADDRESS ________________________________________
_________________________________________________
_________________________________________________
PHONE __________________________________________
COMPUTER ______________________________________
DATE ___________________________________________
APPLICATION FOR LICENSE TO USE VALIDATION SUITE FOR PASCAL

Name and address of requestor: __________________________________________________________

(Company name if requestor is a company) ______________________________________________

Phone Number: ______________________________________________________________________

Name and address to which information should be addressed (write “as above” if the same): ____________________________

Signature of requestor: ____________________________________________________________________________

Date: _________________________________________________________________________________________

In making this application, which should be signed by a responsible person in the case of a company, the requestor agrees that:

a) The Validation Suite is recognized as being the copyrighted, proprietary property of The British Standards Organization and A. H. J. Sale, and

b) The requestor will not distribute or otherwise make available machine-readable copies of the Validation Suite, modified or unmodified, to any third party without written permission of the copyright holders.

In return, the copyright holders grant full permission to use the programs and documentation contained in the Validation Suite for the purpose of compiler validation, acceptance tests, benchmarking, preparation of comparative reports and similar purposes, and to make available the listings of the results of compilation and execution of the programs to third parties in the course of the above activities. In such documents, reference shall be made to the original copyright notice and its source.

Distribution Charge: $300.00

Make checks payable to: Software Consulting Services in US dollars drawn on a US bank.

Remittance must accompany application.

Mail Request and Check To: Software Consulting Services
901 Whittier Dr.
Allentown, PA 18103 USA
Attn: R. J. Cichelli

SOURCE CODE DELIVERY MEDIUM SPECIFICATION

( ) Magnetic tape

9-Track, odd parity, 1/2"x600'. Select Density:

( ) 800 bpi  ( ) 1600 bpi

( ) ANSI-STANDARD. Each logical record is an 80 character card image. Each physical record has a block size of 40 logical records. Select Character Code:

( ) ASCII ( ) EBCDIC

( ) Special DEC System Alternate Formats:

( ) RSX-IAS PIP (requires ANSI MAGtape RSX SYSGEN).

( ) DOS-RSTS FLX.

( ) 8" Diskette

( ) Single Density

( ) Double Density

Format

( ) CP/M  ( ) UCSD III (W. D. Microengine)

( ) UCSD II, IV  ( ) DEC-RT (Single Density)

( ) DEC-RSX Files 11  ( ) IBM 3740 (Single Density EBCDIC)

Special Format

( ) Interleave (1-26)

( ) Skew (0-25)

Office Use Only

Signed: ____________________________________________

Date: _________________________________________________________________________________

Richard J. Cichelli

On Behalf of A. H. J. Sale and B. S. I.
GET MORE FROM YOUR PASCAL SYSTEM
.... JOIN USUS TODAY

USUS is the USER'S GROUP for the most widely used, machine-independent software system.

If you use UCSD Pascal*, Apple Pascal** or the UCSD p-System, USUS will link you with a community of users that share your interests.

USUS was formed to give users an opportunity to promote and influence the development of UCSD Pascal and the UCSD p-System, and to help them learn more about their systems. USUS is non-profit and vendor-independent.

Members get access to the latest UCSD p-System information and to extensive Pascal expertise. In USUS, you have formal and informal opportunities to communicate with and learn from other users via:

• NATIONAL MEETINGS
• USUS NEWS AND REPORT
• ELECTRONIC MAIL
• SOFTWARE LIBRARY
• SPECIAL INTEREST GROUPS

*UCSD Pascal and the UCSD p-System are trademarks of the Regents of the University of California.

**Apple Pascal is a trademark of Apple Computer, Inc.

USUS MEMBERSHIP APPLICATION

(Please complete both sides)

I am applying for $25 individual membership

$500 organization membership

$ __ air mail service surcharge

Rates are for 12 months and cover surface mailing of the newsletter. If you reside outside North America, air mail service is available for a surcharge. It is as follows: $5.00 annually for those in the Caribbean, Central America and Columbia and Venezuela; $10.00 annually for those in South America, Turkey and North Africa; and $15.00 for all others. Check or money order should be drawn on a U. S. bank or U. S. office.

Name/Title ____________________________________________________________
Affiliation ____________________________________________________________
Address ______________________________________________________________

Phone (________) ______________________ TWX/Telex ______________________

Option: Do not print my phone number in USUS rosters
Option: Print only my name and country in USUS rosters
Option: Do not release my name on mailing lists
USUS MEMBERSHIP BENEFITS

* * * * *

• NATIONAL MEETINGS twice a year let you learn from experts and try out the newest products. Meetings feature hardware and software demonstrations, tutorials, technical presentations and information, reduced-cost software library access, special interest group (SIG) meetings, and a chance to query "major" vendors.

• USUS NEWS AND REPORT brings you news and information about your operating system four times a year. It contains technical articles and updates, library catalog listings, SIG reports, a software vendor directory and organizational news.

• ELECTRONIC MAIL puts USUS subscribers in touch with a nationwide network of users. CompuServe MUSUS SIG is for data bases and bulletin board communications. GTE Telemail accommodates one-to-one messages.

• SOFTWARE EXCHANGE LIBRARY offers an extensive collection of tools, games, applications, and aids in UCSD Pascal source code at nominal prices.

• SPECIAL INTEREST GROUPS zero in on specific problems, represent member interests with manufacturers.

For more information, contact: Secretary, USUS, P. O. Box 1148, La Jolla, CA 92038, USA.

Computer System:

___ Z-80 ___ 8080 ___ PDP/LSI-11 ___ 6502/Apple ___ 6800 ___ 6809
___ 9900 ___ 8086/8088 ___ Z8000 ___ 68000 ___ MicroEngine ___ IBM PC

Other ____________________________

I am interested in the following Committees/Special Interest Groups (SIGs):

___ Advanced System Editor SIG ___ Meetings Committee
___ Apple SIG ___ Modula-2 SIG
___ Application Developer’s SIG ___ NEC Advanced PC SIG
___ Communications SIG ___ Publications Committee
___ DEC SIG ___ Sage SIG
___ File Access SIG ___ Software Exchange Library
___ Graphics SIG ___ Technical Issues Committee
___ IBM Display Writer SIG ___ Texas Instruments SIG
___ IBM PC SIG ___ UCSD Pascal Compatability SIG

Mail completed application with check or money order payable to USUS and drawn on a U.S. bank or U.S. office, to Secretary, USUS, P.O. Box 1148, La Jolla, CA 92038, USA.
Membership

Please enter my

☐ New or ☐ Renew

membership in Pascal Users Group. I understand I will receive "Pascal News" whenever it is published in this calendar year.

Pascal News should be mailed
1 yr. ☐ in USA $25 ☐ outside USA $35 ☐ AirMail anywhere $60
3 yr. ☐ in USA $50 ☐ outside USA $80 ☐ AirMail anywhere $125

(Make checks payable to: "Pascal Users Group," drawn on USA bank in US dollars)

Enclosed please find US $______.
on check number ______

(Invoice will be sent on receipt of purchase orders. Payment must be received before newsletter will be sent. Purchase orders will be billed $10 for additional work.)

(I have difficulty reading addresses. Please forgive me and type or print clearly.)

My address is:

NAME

ADDRESS

PHONE

COMPUTER

DATE

☐ This is an address correction here is my old address label:

_________________________________________________________

_________________________________________________________

_________________________________________________________

Attachments: None
JOINING PASCAL USER GROUP?

- Membership is open to anyone: Particularly the Pascal user, teacher, maintainer, implementor, distributor, or just plain fan.
- Please enclose the proper prepayment (check payable to "Pascal User's Group").
- When you join PUG any time within a year: January 1 to December 31, you will receive all issues Pascal News for that year.
- We produce Pascal News as a means toward the end of promoting Pascal and communicating news of events surrounding Pascal to persons interested in Pascal. We are simply interested in the news ourselves and prefer to share it through Pascal News. We desire to minimize paperwork, because we have other work to do.

---

RENEWING?

- Please renew early (before November) and please write us a line or two to tell us what you are doing with Pascal, and tell us what you think of PUG and Pascal News.

ORDERING BACK ISSUES OR EXTRA ISSUES?

- Our unusual policy of automatically sending all issues of Pascal News to anyone who joins within a year means that we eliminate many requests for backissues ahead of time, and we don't have to reprint important information in every issue — especially about Pascal implementations!
- Issues 1 . . . 8 (January, 1974 — May 1977) are out of print.
- Issues 9 . . . 12, 13 . . . 16, & 17 . . . 20, 21 . . . 23 are available from PUG(USA) all for $25.00 a set.
- Extra single copies of new issues (current academic year) are: $10 each — PUG(USA).

SENDING MATERIAL FOR PUBLICATION?

- Your experiences with Pascal (teaching and otherwise), ideas, letters, opinions, notices, news, articles, conference announcements, reports, implementation information, applications, etc. are welcome. Please send material single-spaced and in camera-ready (use a dark ribbon and lines 15.5 cm. wide) form.
- All letters will be printed unless they contain a request to the contrary.
Back Issues

Pascal News
2903 Huntington Road
Cleveland, Ohio 44120

Back issues are requested and sent in sets

$15 [ ] set 0 Issues 1 . . . 8 (January 1974 — May 1977) OUT OF PRINT


$25 [ ] set 2 Issues 13 . . . 16 (December 1978 — October 1979)


Requests from outside USA please add $5 per set.

All memberships entered in 1983 will receive issue 24 and all other issues published in that year.

Make check payable to: "Pascal Users Group," drawn on USA bank in US dollars.

Enclosed please find US $____.____ on check number ________

(I have difficulty reading addresses. Please forgive me and type or print clearly)

My address is:

NAME

ADDRESS

PHONE

COMPUTER

DATE
Back issues are requested and sent in sets

$15 □ set 0 Issues 1 . . . 8 (January 1974 — May 1977) OUT OF PRINT


$25 □ set 2 Issues 13 . . . 16 (December 1978 — October 1979)


Requests from outside USA please add $5 per set.

All memberships entered in 1983 will receive issue 24 and all other issues published in that year. Make check payable to: "Pascal Users Group," drawn on USA bank in US dollars.

Enclosed please find US $____.____ on check number_____

(I have difficulty reading addresses. Please forgive me and type or print clearly)

My address is:

NAME

ADDRESS

PHONE

COMPUTER

DATE
GET MORE FROM YOUR PASCAL SYSTEM
.... JOIN USUS TODAY

USUS is the USER'S GROUP for the most widely used, machine-independent software system.

If you use UCSD Pascal*, Apple Pascal** or the UCSD p-System, USUS will link you with a community of users that share your interests.

USUS was formed to give users an opportunity to promote and influence the development of UCSD Pascal and the UCSD p-System and to help them learn more about their systems. USUS is non-profit and vendor-independent.

Members get access to the latest UCSD p-System information and to extensive Pascal expertise. In USUS, you have formal and informal opportunities to communicate with and learn from other users via:

- NATIONAL MEETINGS
- USUS NEWS AND REPORT
- ELECTRONIC MAIL
- SOFTWARE LIBRARY
- SPECIAL INTEREST GROUPS

*UCSD Pascal and the UCSD p-System are trademarks of the Regents of the University of California.
**Apple Pascal is a trademark of Apple Computer, Inc.

USUS MEMBERSHIP APPLICATION
(Please complete both sides)

I am applying for $25 individual membership
$500 organization membership
$ ______ air mail service surcharge

Rates are for 12 months and cover surface mailing of the newsletter. If you reside outside North America, air mail service is available for a surcharge. It is as follows: $5.00 annually for those in the Caribbean, Central America and Columbia and Venezuela; $10.00 annually for those in South America, Turkey and North Africa; and $15.00 for all others. Check or money order should be drawn on a U.S. bank or U.S. office.

Name/Title ____________________________________________________________
Affiliation ____________________________________________________________
Address ______________________________________________________________
____________________________________________________________________
Phone (_____) _______ - _______ _______ TWX/Telex ________________

Option: Do not print my phone number in USUS rosters
Option: Print only my name and country in USUS rosters
Option: Do not release my name on mailing lists
GET MORE FROM YOUR PASCAL SYSTEM .... JOIN USUS TODAY

USUS is the USER'S GROUP for the most widely used, machine-independent software system.

If you use UCSD Pascal*, Apple Pascal** or the UCSD p-System, USUS will link you with a community of users that share your interests.

USUS was formed to give users an opportunity to promote and influence the development of UCSD Pascal and the UCSD p-System and to help them learn more about their systems. USUS is non-profit and vendor-independent.

Members get access to the latest UCSD p-System information and to extensive Pascal expertise. In USUS, you have formal and informal opportunities to communicate with and learn from other users via:

- NATIONAL MEETINGS
- USUS NEWS AND REPORT
- ELECTRONIC MAIL
- SOFTWARE LIBRARY
- SPECIAL INTEREST GROUPS

*UCSD Pascal and the UCSD p-System are trademarks of the Regents of the University of California.
**Apple Pascal is a trademark of Apple Computer, Inc.

USUS MEMBERSHIP APPLICATION
(Please complete both sides)

I am applying for $25 individual membership

$500 organization membership

$ _____ air mail service surcharge

Rates are for 12 months and cover surface mailing of the newsletter. If you reside outside North America, air mail service is available for a surcharge. It is as follows: $5.00 annually for those in the Caribbean, Central America and Columbia and Venezuela; $10.00 annually for those in South America, Turkey and North Africa; and $15.00 for all others. Check or money order should be drawn on a U. S. bank or U. S. office.

Name/Title ____________________________________________________________

Affiliation ____________________________________________________________

Address ___________________________________________________________________

Phone (______) ___________________________ TWX/Telex ____________________________

Option: Do not print my phone number in USUS rosters

Option: Print only my name and country in USUS rosters

Option: Do not release my name on mailing lists
Please enter my membership in Pascal Users Group. I understand I will receive “Pascal News” whenever it is published in this calendar year.

Pascal News should be mailed
1 yr. □ in USA $25 □ outside USA $35 □ AirMail anywhere $60
3 yr. □ in USA $50 □ outside USA $80 □ AirMail anywhere $125

(Make checks payable to: “Pascal Users Group,” drawn on USA bank in US dollars)

Enclosed please find US $____.
on check number ______

(Invoice will be sent on receipt of purchase orders. Payment must be received before newsletter will be sent. Purchase orders will be billed $10 for additional work.)

(I have difficulty reading addresses. Please forgive me and type or print clearly.)

My address is:

NAME

ADDRESS

PHONE

COMPUTER

DATE

□ This is an address correction here is my old address label:

__________________________________________________________

__________________________________________________________

__________________________________________________________

__________________________________________________________

Coupons
JOINING PASCAL USER GROUP?

• Membership is open to anyone: Particularly the Pascal user, teacher, maintainer, implementor, distributor, or just plain fan.
• Please enclose the proper prepayment (check payable to “Pascal User’s Group”).
• When you join PUG any time within a year: January 1 to December 31, you will receive all issues Pascal News for that year.
• We produce Pascal News as a means toward the end of promoting Pascal and communicating news of events surrounding Pascal to persons interested in Pascal. We are simply interested in the news ourselves and prefer to share it through Pascal News. We desire to minimize paperwork, because we have other work to do.

RENEWING?

• Please renew early (before November) and please write us a line or two to tell us what you are doing with Pascal, and tell us what you think of PUG and Pascal News.

ORDERING BACK ISSUES OR EXTRA ISSUES?

• Our unusual policy of automatically sending all issues of Pascal News to anyone who joins within a year means that we eliminate many requests for back issues ahead of time, and we don’t have to reprint important information in every issue — especially about Pascal implementations!
• Issues 1 . . . 8 (January, 1974 — May 1977) are out of print.
• Issues 9 . . . 12, 13 . . . 16, & 17 . . . 20, 21 . . . 23 are available from PUG(USA) all for $25.00 a set.
• Extra single copies of new issues (current academic year) are: $10 each — PUG(USA).

SENDING MATERIAL FOR PUBLICATION?

• Your experiences with Pascal (teaching and otherwise), ideas, letters, opinions, notices, news, articles, conference announcements, reports, implementation information, applications, etc. are welcome. Please send material single-spaced and in camera-ready (use a dark ribbon and lines 15.5 cm. wide) form.
• All letters will be printed unless they contain a request to the contrary.
Facts about Pascal, THE PROGRAMMING LANGUAGE:

Pascal is a small, practical, and general-purpose (but not all-purpose) programming language possessing algorithmic and data structures to aid systematic programming. Pascal was intended to be easy to learn and read by humans, and efficient to translate by computers.

Pascal has met these goals and is being used successfully for:

- teaching programming concepts
- developing reliable "production" software
- implementing software efficiently on today's machines
- writing portable software

Pascal implementations exist for more than 105 different computer systems, and this number increases every month. The "Implementation Notes" section of Pascal News describes how to obtain them.

The standard reference ISO 7185 tutorial manual for Pascal is:


Introductory textbooks about Pascal are described in the "Here and There" section of Pascal News.

The programming language, Pascal, was named after the mathematician and religious fanatic Blaise Pascal (1623-1662). Pascal is not an acronym.

Remember, Pascal User's Group is each individual member's group. We currently have more than 3500 active members in more than 41 countries.
This is your last issue if you have not renewed for 1983!