

ABM \$10.00

PASCAL USERS GROUP

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

Number

26

JULY 83

POLICY: PASCAL NEWS

(Jan. 83)

- *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 Srully 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

47 IMPLEMENTATION REPORT

VALIDATION SUITE REPORT

48 OmegaSoft Pascal Version 2

51 SUBSCRIPTION COUPON

53 VALIDATION SUITE COUPON

55 USUS MEMBERSHIP COUPON

Charles Gaffney Publisher and Editor

The Pascal Newsletter is published by the Pascal Users Group, 2903 Huntington Rd., Cleveland, Ohio 44120. The Pascal Newsletter is a direct benefit of membership in PUG.

Membership dues in PUG are \$25.00 US regular, other forms of membership please inquire. Inquiries regarding membership should be sent to the above address. Newsletter correspondence and advertising should be sent to the editor at the aforementioned address.

Advertising Rates: \$300.00 Full Page. Please give your preference of magazine location: front, center, or back.

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 micro-computer 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. Sully 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 "Referencer", 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 Tandenbaum, 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 (misabeled #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!

Open Forum

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 our #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

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 help 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 text book 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."

One of the letters mentions high resolution graphics. I know of two texts that use Pascal as the illustrative language of their algorithms. They are "Principles of Interactive Computer Graphics" by Williams Newman and Robert Sproull, 1979 McGraw-Hill and "Fundamentals of Interactive Computer Graphics" by James Foley and Andries Van Dam, 1982 Addison-Wesley.

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

Open Forum Open Forum

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

Dear Sir,

I just received issue number 25 of *Pascal News* and was surprised to find an implementation note for our

Open Forum

Pascal compiler. What makes it surprising is that to the best of my knowledge I have never sent in an entry, and the information provided is about a year and a half out of date.

In case you would like to provide your readers with valid information, I have enclosed an implementation note for the currently available compiler. I have also enclosed a copy of the ISO validation suite report from our language manual.

Work is currently being done on moving this compiler to the 68000 family of processors and should be available by the end of 1983.

On another note, I have received issues number 21, 22/23, and 25, but not issue 24. I am also enclosing a check for a 3 year membership — please see if you can determine what happened to number 24.

Sincerely,
Robert Reimiller
Owner, OmegaSoft
5787 Brandywine Ct.
Camarillo, CA 93010

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 our *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

November 30, 1982

Pascal is being used for process control of laser trimming systems. We use Oregon Software Pascal.

Barbara Huseby, Training Dept.
Electro Scientific Industries
13900 N.W. Science Park Drive
Portland, OR 97229

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

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 x 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

Open Forum

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 tht 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 III 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 paper-work (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.
Villanova University
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	ERA-50 Computer (8-bit, 8085 base)
Pascal MT+ under CP/M and MP/M	ERA-50 Computer (8-bit, 8085 base) ERA-50 Computer (8-bit, 8085 base)
Pascal MT+ 86 under CP/M-86 and MP/M-86	ERA-80 Computer (16 bit, 8086/8087 based) ERA-80 Computer (16 bit, 8086/8087 based)

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

1-8-82

Dear Sir/Madam,

This is the first letter I write to contact you. Let
Open Forum

me introduce myself first. I am a student pursuing a computer course in the Hong Kong Polytechnic — a licensed user of your OMSI-PASCAL-2 V1.2. I don't know what your definition of user may be. May it be my Polytechnic or any student or programmer who use your OMSI PASCAL-1 under the Polytechnic, I venture to call myself a user in this letter, and would like to join the Pascal Users' Group and receive the newsletter.

In the past few months, I have been doing extensive programming using PASCAL, and find it very handy, especially in writing structured programs. However, until recently when I develop some system programs, I find problems. I discover that there is no source listing or documentation on the OMSI PASCAL-1 run time system (possibly in file FPP.RTS) and its relationship with RSTS/E, and I cannot interface with the low level I/O trap handlers without knowing their details. I find some problems on the RESET ODT mode, but I cannot deal with it in assembly level.

All in all, my problem is highly personal and does not in any way bear relation with the Hong Kong Polytechnic. However, as a student on computing, I don't want to leave problem unsolved. So, please send me any informational help, if possible.

Included please find a bank draft of \$6 for subscription.

May I state once more my request. I need information on OMSI PASCAL-1 run time system especially the EMT trap handling.

Thank you very much in advance.

Yours faithfully,

Mr Kam Man-Kai
Flat 8
3/F Ting Yin House
Siu On Court
Tuen Mun - N.T.
Hong Kong

6th November, 1982.

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 J0P 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 {
4 * Purpose:
5   This program is an implementation of APL in Pascal.
6
7 * Authors:
8   Vincent Dichristofano
9   Alan Kaniss
10  Thomas Robinson
11  John Santini
12   authors' affiliation - NADC
13
14   Phil. PA. USA
15
16   project leader: Dr. Joseph Mezzaroba
17
18   This program was written as part of an independent study
19   course at Villanova University.
20
21 * Submitted and accepted for Pascal News. DEC 1976.
22
23 }
24 Label
25 100;
26
27 const
28   prefix1 = 60;
29   prefix2 = 62 { prefix for CDC ASCII 12-bit codes };
30   MaxVarNameLength = 10;
31   MaxInputLine = 132;
32   InputArraySize = 134;
33   NumberOfMessages = 100;
34   MessageLength = 80;
35
36 type
37   PackedString = packed array [1 .. MaxVarNameLength] of 0 .. 8191;
38   TokenNoun =
39     (FormRes, FormArg, GlobVar, MonadOper, ReductOper, DyadOper,
40     SpecOper, constant, StatEnd);
41   values = record
42     RealVal: real;
43     NextValue: ^values
44   end;
45   VarTab = record
46     VarName: PackedString [ v1 ];
47     FuncTabPtr: ^FuncTab [ v2 - ftab ];
48     ValTabPtr: ^ValTab [ v3 - vtab ];
49     DeferredValTabPtr: ^FParamTab;
50     NextVarTabPtr: ^VarTab
51   end;
52   ValTab = record
53     IntermedResult: Boolean;
54     dimensions: integer;
55     FirstDimen: ^DimenInfo;
56     ForwardOrder: Boolean;
57     FirstValue: ^values;
58     NextValTabLink: ^ValTab
59   end;
60   TokenTable = record
61     NextToken: ^TokenTable;
62     case noun: TokenNoun of [ p ]
63       FormRes, FormArg, GlobVar: { vtab }
64         (VarTabPtr: ^VarTab);
65       MonadOper: (NonIndex: integer);
66       ReductOper: (RedInIdx: integer);
67       DyadOper: (DOPInIdx: integer);
68       SpecOper: (CharInIdx: integer);
69       constant: (ValTabPtr: ^ValTab);
70       StatEnd: (EndAdj: integer)
71     end;
72   vfunc = record
73     NextStant: ^TokenTable;
74     NextVFuncPtr: ^vfunc;
75     StatLabel: PackedString
76   end;
77   OperatorType = (niladic, monadic, dyadic);
78   FuncTab = record
79     FuncName: PackedString [ f1 ];
80     arity: OperatorType [ f2 ];
81     result: Boolean [ f3 true = explicit ];
82     ResultName: PackedString [ f4 ];
83     LeftArg: PackedString [ f5 ];
84     RightArg: PackedString [ f6 ];
85     FirstStatement: ^vfunc;
86     NextFuncTabPtr: ^FuncTab;
87     NumOfStatements: integer
88   end;
89   FParamTab = record
90     PtrVal: ^ValTab [ sd1 and sd2 ];
91     LastParm: ^FParamTab [ link to last ]
92   [ sd1 or sd2 ]
93   end;
94   DimenInfo = record
95     NextDimen: ^DimenInfo;
96     dimenlength: integer
97   end;
98   OpRecord = record
99     OpIndex: integer;
100    OpSymbol: integer
101  end;
102  OperandTab = record
103    OperPtr: ^ValTab [ sval ];
104    LastOper: ^OperandTab [ link to last sval ]
105  end;
106  SubrTab = record [ sf ]
107    CalledSubr: ^FuncTab [ s1 ];
108    TokenCallingSubr: ^TokenTable [ s2 ];
109    StateCallingSubr: ^vfunc [ s3 ];
110    LastSubrPtr: ^SubrTab [ link to last sf ]
111  end;
112  OpTable = array [1 .. 16] of OpRecord;
113  VarTabPtrType = ^VarTab;
114  TypeValTabPtr = ^ValTab;
115  TokenPtr = ^TokenTable;
116  PtrFuncTab = ^FuncTab;
117  TypeValuesPtr = ^values;
118  APLcharSet =
119    (Asymbol, Bsymbol, CSymbol, DSymbol, ESymbol, FSymbol, GSymbol,
120    HSymbol, ISymbol, JSymbol, KSymbol, LSymbol, MSymbol, NSymbol,
121    OSymbol, PSymbol, QSymbol, RSymbol, SSymbol, TSymbol, USymbol,
122    VSymbol, WSymbol, XSymbol, YSymbol, ZSymbol, OneSymbol, TwoSymbol,
123    ThreeSymbol, FourSymbol, FiveSymbol, SixSymbol, SevenSymbol,
124    EightSymbol, NineSymbol, ZeroSymbol, colon, RightArrow, LeftArrow,
125    SmallCircle, period, LeftParen, RightParen, LeftBracket,
126    RightBracket, semicolon, quadrangle, space, plus, minus, times,
127    divide, asterisk, iota, rho, comma, tilde, equals, NotEqual,
128    LessThan, LessOrEqual, GreaterOrEqual, GreaterThan, AndSymbol,
129    OrSymbol, ceiling, floor, LargeCircle, ForwardSlash, DoubleQuote,
130    negative, QuestionMark, omega, epsilon, UpArrow, DownArrow, alpha,
131    UnderScore, del, delta, SingleQuote, EastCap, WestCap, SouthCap,
132    NorthCap, ibeam, TBeam, VerticalStroke, BackwardSlash);
133
134   text = file of char;
135
136 var
137   XColonSym, XRightArrow, XLeftArrow, XLittleCircle, XPeriod, XLeftPar,
138   XRightPar, XLeftBracket, XRightBracket, XSemicolon, XQuadSym:
139   integer;
140   character: array [APLcharSet] of integer;
141   APLstatement: array [1 .. InputArraySize] of integer;
142   digits: array [OneSymbol .. ZeroSymbol] of integer;
143   ErrorMessage: packed array [1 .. NumberOfMessages, 1 .. MessageLength] of
144   char;
145   APLfile: text;
146   MOPTab, DOPTab, RedTab, CharTab, SpecTab: OpTable;
147   SaveLabel: PackedString;
148   name: PackedString;
149   NewTokenPtr, OldTokenPtr, HoldTokenPtr, SaveTokenPtr: ^TokenTable;
150   TestFuncPtr, NewFuncTabPtr, OldFuncTabPtr: ^FuncTab;
151   NewVarTabPtr, OldVarTabPtr: ^VarTab;
152   LeftValPtr, RightValPtr, ValPtr: ^values;
153   NewValues, NewValPtr: ^values;
154   NewDim: ^DimenInfo;
155   DimPtr, NewPtr, LeftDimPtr, RightDimPtr: ^DimenInfo;
156   VarPointer: ^VarTab;
157   OldVFuncPtr, NewVFuncPtr: ^vfunc;
158   NewValTabLink, OldValTabLink: ^ValTab;
159   position: integer;
160   LineLength: integer;
161   code, ColCnt: integer;
162   FuncStatements: integer;
163   TokenError, FirstFunction: Boolean;
164   LineTooLong, HasLabel: Boolean;
165   switch, FunctionMode, TokenSwitch, ItsAnIdentifier: Boolean;
166   OperTabPtr: ^OperandTab [ sv ];

```

```

167 PtrLastOper := "OperandTab;
168 SubrTabPtr := "SubrTab;
169 RParamPtr := "FParamTab | p1 |;
170 LParamPtr := "FParamTab | p2 |;
171 VFuncPtr := "vfunc | nl |;
172 hold := "TokenTable | holds last symbol |;
173
174
175 procedure InitParser;
176
177 begin
178   OperTabPtr := nil; SubrTabPtr := nil; LParamPtr := nil;
179   RParamPtr := nil; VFuncPtr := nil; hold := nil; XColonSym := 1;
180   XRightArrow := 2; XLeftArrow := 3; XLittleCircle := 4;
181   XPeriod := 5; XLeftPar := 6; XRightPar := 7;
182   XLeftBracket := 8; XRightBracket := 9; XSemicolon := 10;
183   XQuadSym := 11; new(OperTabPtr); OperTabPtr.LastOper := nil;
184   PtrLastOper := OperTabPtr;
185 end { initparser };
186
187
188 procedure InitializeCharacterSet
189   { read installation character set from file };
190
191 var
192   TestForPrefix: integer;
193   FileCharacter: char;
194   SymbolIndex: APLcharSet;
195
196 begin
197   reset(APLfile);
198   for SymbolIndex := a symbol to BackwardSlash do
199     begin
200       read(APLfile, FileCharacter);
201
202       { The following code would be removed for non-CDC installations }
203
204       TestForPrefix := ord(FileCharacter);
205       if (TestForPrefix = prefix1) or (TestForPrefix = prefix2)
206       then
207         begin
208           read(APLfile, FileCharacter);
209           character[SymbolIndex] := 100 * TestForPrefix + ord(
210             FileCharacter);
211         end
212       else
213         character[SymbolIndex] := ord(FileCharacter)
214       end
215     end
216   end { initializecharacter set };
217
218
219 procedure ReadInErrorMsgs;
220
221 var
222   MsgRow, MsgCol: integer;
223
224 begin
225   readln(APLfile);
226   for MsgRow := 1 to NumberOfMessages do
227     for MsgCol := 1 to MessageLength do
228       ErrorMsgs[MsgRow, MsgCol] := ' ' { blank out error messages };
229     for MsgRow := 1 to NumberOfMessages do
230       begin { read in error messages from file }
231         MsgCol := 0;
232         while not eoln(APLfile) do
233           begin
234             MsgCol := MsgCol + 1;
235             read(APLfile, ErrorMsgs[MsgRow, MsgCol]);
236           end;
237         readln(APLfile);
238       end
239     end { readinerrormsgs };
240
241
242 procedure FillUpTables;
243
244 begin
245   {
246     monadic operators
247   }
248   MOpTab[1].OpSymbol := character[plus]; MOpTab[1].OpIndex := 2;
249   MOpTab[2].OpSymbol := character[minus]; MOpTab[2].OpIndex := 3;
250   MOpTab[3].OpSymbol := character[times]; MOpTab[3].OpIndex := 4;
251   MOpTab[4].OpSymbol := character[divide]; MOpTab[4].OpIndex := 5;
252   MOpTab[5].OpSymbol := character[asterisk]; MOpTab[5].OpIndex := 6;
253   MOpTab[6].OpSymbol := character[iota]; MOpTab[6].OpIndex := 21;
254   MOpTab[7].OpSymbol := character[rho]; MOpTab[7].OpIndex := 22;
255   MOpTab[8].OpSymbol := character[comma]; MOpTab[8].OpIndex := 23;
256   MOpTab[9].OpSymbol := character[tilde]; MOpTab[9].OpIndex := 1;
257
258   {
259     dyadic operators
260   }
261   DOpTab[1].OpSymbol := character[plus]; DOpTab[1].OpIndex := 52;
262   DOpTab[2].OpSymbol := character[minus]; DOpTab[2].OpIndex := 53;
263   DOpTab[3].OpSymbol := character[times]; DOpTab[3].OpIndex := 54;
264   DOpTab[4].OpSymbol := character[divide]; DOpTab[4].OpIndex := 55;
265   DOpTab[5].OpSymbol := character[asterisk];
266   DOpTab[5].OpIndex := 56; DOpTab[6].OpSymbol := character[iota];
267   DOpTab[6].OpIndex := 87; DOpTab[7].OpSymbol := character[rho];
268   DOpTab[7].OpIndex := 88; DOpTab[8].OpSymbol := character[comma];
269
270   DOpTab[8].OpIndex := 89; DOpTab[9].OpSymbol := character[equals];
271   DOpTab[9].OpIndex := 71;
272   DOpTab[10].OpSymbol := character[notEqual];
273   DOpTab[10].OpIndex := 72;
274   DOpTab[11].OpSymbol := character[lessThne];
275   DOpTab[11].OpIndex := 73;
276   DOpTab[12].OpSymbol := character[lessOrEqual];
277   DOpTab[12].OpIndex := 74;
278   DOpTab[13].OpSymbol := character[greaterOrEqual];
279   DOpTab[13].OpIndex := 75;
280   DOpTab[14].OpSymbol := character[greaterThan];
281   DOpTab[14].OpIndex := 76;
282   DOpTab[15].OpSymbol := character[andSymbol];
283   DOpTab[15].OpIndex := 77;
284   DOpTab[16].OpSymbol := character[orSymbol];
285   DOpTab[16].OpIndex := 78;
286
287   {
288     special character
289   }
290   CharTab[1].OpSymbol := character[colon];
291   CharTab[2].OpSymbol := character[rightArrow];
292   CharTab[3].OpSymbol := character[leftArrow];
293   CharTab[4].OpSymbol := character[smallCircle];
294   CharTab[5].OpSymbol := character[period];
295   CharTab[6].OpSymbol := character[leftParen];
296   CharTab[7].OpSymbol := character[rightParen];
297   CharTab[8].OpSymbol := character[leftBracket];
298   CharTab[9].OpSymbol := character[rightBracket];
299   CharTab[10].OpSymbol := character[semicolon];
300   CharTab[11].OpSymbol := character[quadrangle];
301   CharTab[12].OpSymbol := character[space];
302   Spectab[1].OpSymbol := character[colon];
303   Spectab[2].OpSymbol := character[rightArrow];
304   Spectab[3].OpSymbol := character[leftArrow];
305   Spectab[4].OpSymbol := character[leftParen];
306   Spectab[5].OpSymbol := character[semicolon];
307   Spectab[6].OpSymbol := character[leftBracket];
308
309   {
310     reduction operator
311   }
312   RedTab[1].OpSymbol := character[plus]; RedTab[1].OpIndex := 2;
313   RedTab[2].OpSymbol := character[minus]; RedTab[2].OpIndex := 3;
314   RedTab[3].OpSymbol := character[times]; RedTab[3].OpIndex := 4;
315   RedTab[4].OpSymbol := character[divide]; RedTab[4].OpIndex := 5;
316   RedTab[5].OpSymbol := character[asterisk]; RedTab[5].OpIndex := 6;
317   RedTab[6].OpSymbol := character[equals]; RedTab[6].OpIndex := 21;
318   RedTab[7].OpSymbol := character[notEqual];
319   RedTab[7].OpIndex := 22;
320   RedTab[8].OpSymbol := character[lessThne];
321   RedTab[8].OpIndex := 23;
322   RedTab[9].OpSymbol := character[lessOrEqual];
323   RedTab[9].OpIndex := 24;
324   RedTab[10].OpSymbol := character[greaterOrEqual];
325   RedTab[10].OpIndex := 25;
326   RedTab[11].OpSymbol := character[greaterThan];
327   RedTab[11].OpIndex := 26;
328   RedTab[12].OpSymbol := character[andSymbol];
329   RedTab[12].OpIndex := 27;
330   RedTab[13].OpSymbol := character[orSymbol];
331   RedTab[13].OpIndex := 28;
332   RedTab[14].OpSymbol := character[ceiling];
333   RedTab[14].OpIndex := 29; RedTab[15].OpSymbol := character[floor];
334   RedTab[15].OpIndex := 30;
335   RedTab[16].OpSymbol := character[largeCircle];
336   RedTab[16].OpIndex := 31; digits[OneSymbol] := 1;
337   digits[TwoSymbol] := 2; digits[ThreeSymbol] := 3;
338   digits[FourSymbol] := 4; digits[FiveSymbol] := 5;
339   digits[SixSymbol] := 6; digits[SevenSymbol] := 7;
340   digits[EightSymbol] := 8; digits[NineSymbol] := 9;
341   digits[ZeroSymbol] := 0;
342 end { filluptables };
343
344 procedure PrintAPLStatement;
345
346 var
347   prefix, num: integer;
348   index: integer;
349
350 begin
351   for index := 1 to LineLength do
352     begin
353       if APLstatement[index] > 6000
354       then
355         begin
356           prefix := APLstatement[index] div 100; write(chr(prefix));
357           num := APLstatement[index] - 100 * prefix;
358           write(chr(num));
359         end
360       else write(chr(APLstatement[index]))
361       end;
362     end { printapstatement };
363
364 procedure SError(ErrorIndex: integer);
365
366 var
367   MsgCol: integer;
368
369 begin
370

```

```

371 TokenError := true;
372 for MsgCol := 1 to MessageLength do
373   write(ErrorMsgs[ErrorIndex, MsgCol]);
374 writeln; PrintAPLstatement ( echo statement to user );
375 for MsgCol := 1 to (position - 1) do write(' ');
376 writeln(chr(character[UpArrow])) | print pointer to user error );
377 end | error );
378
379
380 procedure SkipSpaces;
381
382 begin
383   while (APLstatement[position] = character[space]) and (position <=
384     LineLength) do
385     position := position + 1
386   end | skipspaces );
387
388
389 procedure GetAPLstatement;
390
391 var
392   InputChar: char;
393   TestForPrefix: integer;
394   FirstTry: Boolean;
395
396 begin
397   for LineLength := 1 to MaxInputLine do
398     APLstatement[LineLength] := character[space] | blank out line );
399   LineLength := 0; FirstTry := true; position := 1;
400   LineTooLong := false;
401   APLstatement[InputArraySize] := character[omega];
402   APLstatement[InputArraySize - 1] := character[space]
403   | set end-of-line );
404   repeat
405     begin
406       if not FirstTry then getseg(input) | test for "or" only );
407       FirstTry := false;
408       while (not eoln(input)) and (not LineTooLong) do
409         if LineLength < MaxInputLine
410           then
411             begin
412               LineLength := LineLength + 1; read(InputChar);
413             ( The following code would be removed for non-CDC installations )
414             TestForPrefix := ord(InputChar);
415             if (TestForPrefix = prefix1) or (TestForPrefix = prefix2)
416               then
417                 begin
418                   read(InputChar);
419                   APLstatement[LineLength] := 100 * TestForPrefix + ord(
420                     InputChar);
421                 end
422               else
423                 (
424                   )
425                 APLstatement[LineLength] := ord(InputChar)
426               end
427             else LineTooLong := true
428           end
429         until LineLength <> 0 | reject null lines );
430         if LineTooLong then SError(71)
431       end | getaplstatement );
432
433
434
435 function ItsADigit(TestChar: integer): Boolean;
436
437 var
438   DigitIndex: APLcharSet;
439
440 begin | test to see if input character is a digit )
441   ItsADigit := false;
442   for DigitIndex := OneSymbol to ZeroSymbol do
443     if TestChar = character[DigitIndex] then ItsADigit := true
444   end | itsadigit );
445
446
447
448 function ItsALetter(TestChar: integer): Boolean;
449
450 var
451   LetterIndex: APLcharSet;
452
453 begin | test to see if input character is a letter )
454   ItsALetter := false;
455   for LetterIndex := asymbol to Zsymbol do
456     if TestChar = character[LetterIndex] then ItsALetter := true
457   end | itsaletter );
458
459
460
461 function CharToNum(TestChar: integer): integer;
462
463 var
464   DigitIndex: APLcharSet;
465
466 begin | chage a character to a number )
467   for DigitIndex := OneSymbol to ZeroSymbol do
468     if TestChar = character[DigitIndex]
469       then CharToNum := digits[DigitIndex]
470     end | chartonum );
471
472
473
474 function NamesMatch(NameOne, NameToo: PackedString): Boolean;
475
476 var
477   Index: integer;
478
479 begin | see if two names (identifiers) are the same )
480   NamesMatch := true;
481   for index := 1 to MaxVarNameLength do
482     if NameOne[index] <> NameToo[index] then NamesMatch := false
483   end | namesmatch );
484
485
486
487 procedure TableLookUp(TestChar, TableLength: integer; table: OpTable;
488   var TableIndex: integer);
489
490 var
491   Index: integer;
492
493 begin | check for membership in a given table )
494   TableIndex := 0;
495   for index := 1 to TableLength do
496     if TestChar = table[index].OpSymbol then TableIndex := index
497   end | tablelookup );
498
499
500
501 procedure identifier(var name: PackedString; var ItsAnIdentifier:
502   Boolean);
503
504 var
505   NameLength: integer;
506   NameTooLong: Boolean;
507
508 begin
509   ItsAnIdentifier := false; SkipSpaces;
510   if ItsALetter(APLstatement[position])
511     then
512       begin
513         NameTooLong := false; ItsAnIdentifier := true;
514         for NameLength := 1 to MaxVarNameLength do | blank out name )
515           name[NameLength] := character[space];
516         NameLength := 0;
517         while (ItsALetter(APLstatement[position]) or (ItsADigit(
518           APLstatement[position])) do
519           begin | build identifier )
520             NameLength := NameLength + 1;
521             if NameLength <= MaxVarNameLength
522               then name[NameLength] := APLstatement[position]
523             else NameTooLong := true;
524             position := position + 1
525           end;
526           if NameTooLong
527             then SError(70) | name greater than maxlength )
528           end
529         end | identifier );
530
531
532
533 procedure MakeNumber(var RealNumber: real; var ItsANumber: Boolean);
534
535 var
536   sign, DigitCount: integer;
537
538 begin | convert character input string to numerical representation )
539   ItsANumber := false; SkipSpaces; sign := 1; DigitCount := 0;
540   RealNumber := 0.0;
541   if (APLstatement[position] = character[negative]) or (ItsADigit(
542     APLstatement[position]))
543     then
544       begin
545         ItsANumber := true;
546         if APLstatement[position] = character[negative]
547           then begin sign := - 1; position := position + 1 end;
548         if not ItsADigit(APLstatement[position])
549           then
550             begin
551               SError(1) | digit must follow a minus sign );
552             ItsANumber := false;
553           end
554         else
555           begin | form whole number portion )
556             while ItsADigit(APLstatement[position]) do
557               begin
558                 RealNumber := 10.0 * RealNumber + CharToNum(APLstatement
559                   [position]);
560                 position := position + 1
561               end;
562               if APLstatement[position] = character[period]
563                 then
564                   begin
565                     position := position + 1;
566                     while ItsADigit(APLstatement[position]) do
567                       begin | form fractional portion )
568                         RealNumber := RealNumber + CharToNum(APLstatement[
569                           position]) * exp((- 1.0 - DigitCount) * 2.3025851
570                           );
571                         DigitCount := DigitCount + 1;
572                         position := position + 1;
573                       end;
574                       if DigitCount = 0 then
575                         begin
576                           SError(2) | digits must follow a decimal point );
577                         ItsANumber := false;
578                       end
579                     end;
580                   end;
581                 end | identifier );
582
583
584

```

```

575     RealNumber := RealNumber * sign
576     end
577   end
578   end { makeanumber };
579
580 function MonadicReference: Boolean;
581
582   var
583     SubPosition, TableIndex: integer;
584
585   begin { see if operator is monadic within context of input line }
586     MonadicReference := false;
587     if NewTokenPtr^.NextToken^.noun = StatEnd
588     then MonadicReference := true
589     else
590       begin
591         SubPosition := position - 1;
592         while (SubPosition > 0) and (APLstatement[SubPosition] =
593           character[space]) do
594           SubPosition := SubPosition - 1 { get last non-blank };
595         if SubPosition <> 0 then
596           TableLookup(APLstatement[SubPosition], 6, SpecTab, TableIndex)
597         ;
598         if (TableIndex <> 0) or (SubPosition = 0)
599         then MonadicReference := true
600         else
601           if (NewTokenPtr^.NextToken^.noun <> FormRes) and (NewTokenPtr
602             ^.NextToken^.noun <> FormArg) and (NewTokenPtr^.NextToken
603             ^.noun <> GlobVar) and (NewTokenPtr^.NextToken^.noun <>
604             constant) and (APLstatement[SubPosition] <> character[
605             period]) and (APLstatement[SubPosition] <> character[
606             RightParen]) and (APLstatement[SubPosition] <> character[
607             RightBracket])
608           then MonadicReference := true
609           end
610         end { monadicreference };
611       end
612     end
613   procedure DyadicOpCheck;
614
615     var
616       TableIndex: integer;
617
618     begin
619       TableLookup(APLstatement[position], 16, DOpTab, TableIndex);
620       if TableIndex = 0
621       then
622         begin
623           TableLookup(APLstatement[position], 12, CharTab, TableIndex);
624           if TableIndex = 0
625           then
626             if APLstatement[position] = character[SouthCap]
627             then
628               begin
629                 OldTokenPtr := SaveTokenPtr; dispose(NewTokenPtr);
630                 NewTokenPtr := SaveTokenPtr; position := LineLength + 1;
631                 end { this was a comment - ignore remainder of line }
632               else SError(4) { invalid character encountered }
633               else
634                 begin { special character encountered }
635                   NewTokenPtr^.noun := SpecOper;
636                   NewTokenPtr^.CharIndx := TableIndex
637                 end
638               end
639             else
640               if MonadicReference
641               then SError(74) { monadic reference to dyadic operator }
642               else
643                 begin { operator is dyadic }
644                   NewTokenPtr^.noun := DyadOper;
645                   NewTokenPtr^.DOpIndx := TableIndex
646                 end
647               end { dyadicopcheck };
648             end
649           end
650         procedure CheckOtherTables;
651
652           var
653             TableIndex: integer;
654             ChkIndex: integer;
655
656           function NextNonBlank: integer;
657
658             begin
659               ChkIndex := position + 1;
660               while (ChkIndex < LineLength) and (APLstatement[ChkIndex] =
661                 character[space]) do
662                 ChkIndex := ChkIndex + 1;
663               NextNonBlank := APLstatement[ChkIndex];
664             end { nextnonblank };
665
666           begin { checkothertables }
667             if NextNonBlank = character[forwardSlash]
668             then
669               begin
670                 TableLookup(APLstatement[position], 16, RedTab, TableIndex);
671                 if TableIndex = 0
672                 then SError(72) { invalid reduction operator }
673                 else
674                   if not MonadicReference
675                   then SError(73) { dyadic reduction reference }
676                   else
677                     begin { operator is valid reduction operator }
678                       NewTokenPtr^.noun := ReductOper;
679                       NewTokenPtr^.RedIndx := TableIndex;
680                       end;
681                     position := ChkIndex + 1;
682                   end
683                 else
684                   begin
685                     TableLookup(APLstatement[position], 9, MOpTab, TableIndex);
686                     if TableIndex = 0 then DyadicOpCheck
687                     else
688                       if not MonadicReference then DyadicOpCheck
689                       else
690                         begin { operator is monadic }
691                           NewTokenPtr^.noun := MonadOper;
692                           NewTokenPtr^.MonIndex := TableIndex;
693                           end;
694                         position := position + 1;
695                       end
696                     end { checkothertables };
697                   end
698                 procedure TryToGetANumber;
699
700                   var
701                     NumberCount: integer;
702                     RealNumber: real;
703                     ItsANumber: Boolean;
704
705                   begin
706                     NumberCount := 0; MakeNumber(RealNumber, ItsANumber);
707                     if not ItsANumber then CheckOtherTables
708                     else
709                       begin { store values in value table }
710                         new(NewValTabLink);
711                         NewValTabLink^.NextValTabLink := OldValTabLink;
712                         OldValTabLink := NewValTabLink;
713                         NewValTabLink^.ForwardOrder := true;
714                         if FunctionMode then NewValTabLink^.IntermedResult := false
715                         else NewValTabLink^.IntermedResult := true;
716                         switch := true;
717                         while ItsANumber do
718                           begin
719                             NumberCount := NumberCount + 1; new(NewValues);
720                             if switch
721                             then
722                               begin
723                                 switch := false;
724                                 NewValTabLink^.FirstValue := NewValues
725                               end
726                             else NewValPtr^.NextValue := NewValues;
727                             NewValues^.RealVal := RealNumber; NewValPtr := NewValues;
728                             MakeNumber(RealNumber, ItsANumber)
729                             end;
730                           NewValues^.NextValue := nil;
731                           if NumberCount > 1
732                           then
733                             begin
734                               NewValTabLink^.dimensions := 1 { number is a vector };
735                               new(NewDim); NewValTabLink^.FirstDimen := NewDim;
736                               NewDim^.dimenlength := NumberCount;
737                               NewDim^.NextDimen := nil
738                             end
739                           else
740                             begin
741                               NewValTabLink^.dimensions := 0 { number is a scalar };
742                               NewValTabLink^.FirstDimen := nil
743                             end;
744                           NewTokenPtr^.noun := constant;
745                           NewTokenPtr^.ValTabPtr := NewValTabLink;
746                           end
747                         end { trytogetanumber };
748                       end
749                     function NameInVarTable(name: PackedString; var VarPointer:
750                       VarTabPtrType; TestFuncPtr: PtrFuncTab): Boolean;
751
752                       var
753                         Found: Boolean;
754
755                       begin
756                         found := false; VarPointer := OldVarTabPtr;
757                         while (VarPointer <> nil) and (not found) do
758                           begin
759                             if (NamesMatch(name, VarPointer^.VarName) and (VarPointer^.
760                               FuncTabPtr = TestFuncPtr) { test for global var }
761                             then found := true
762                             else VarPointer := VarPointer^.NextVarTabPtr
763                             end;
764                           NameInVarTable := found;
765                           end { nameinvartable };
766                       end
767                     procedure AddNameToVarTable(name: PackedString);
768
769                       begin { new variable name encountered }
770                         new(NewVarTabPtr); NewVarTabPtr^.NextVarTabPtr := OldVarTabPtr;
771                         OldVarTabPtr := NewVarTabPtr; NewVarTabPtr^.VarName := name;
772                         NewVarTabPtr^.ValTabPtr := nil;
773                       end

```

```

779 if NewTokenPtr <> nil
780 then
781   if (NewTokenPtr^.noun = FormRes) or (NewTokenPtr^.noun = FormArg
782   )
783   then NewVarTabPtr^.FuncTabPtr := NewFuncTabPtr
784   else NewVarTabPtr^.FuncTabPtr := nil
785   end { addnametovariable };
786
787
788 function FunctionAlreadyDefined(var NewFuncName: PackedString; var
789   FuncIndex: PtrFuncTab): Boolean;
790
791 var
792   found: Boolean;
793
794 begin
795   found := false; FuncIndex := OldFuncTabPtr;
796   while (FuncIndex <> nil) and (not found) and (NewFuncTabPtr <>
797   nil) do
798     if NamesMatch(FuncIndex^.FuncName, NewFuncName)
799     then found := true
800     else FuncIndex := FuncIndex^.NextFuncTabPtr;
801     FunctionAlreadyDefined := found
802   end { functionalreadydefined };
803
804
805 procedure MakeTokenLink;
806
807 begin
808   new(NewTokenPtr); NewTokenPtr^.NextToken := OldTokenPtr;
809   SaveTokenPtr := OldTokenPtr; OldTokenPtr := NewTokenPtr
810 end { maketokenlink };
811
812
813 procedure ProcessFunctionHeader;
814
815 var
816   DummyPtr: ^FuncTab;
817   name1, name2, name3: PackedString;
818   ItsAnIdentifier, FuncHeadError: Boolean;
819   ArityIndex: integer;
820
821 begin
822   FuncHeadError := false; FunctionMode := true;
823   FuncStatements := - 1;
824   if FirstFunction
825   then begin FuncStatements := 0; FirstFunction := false; end;
826   ArityIndex := 1; position := position + 1;
827   identifier(name1, ItsAnIdentifier);
828   if not ItsAnIdentifier
829   then
830     begin
831       SError(7) { unrecognizable function/argument name };
832       FunctionMode := false { exit function mode };
833       FuncHeadError := true
834     end
835   else
836     begin
837       new(NewFuncTabPtr); SkipSpaces;
838       if APLstatement[position] = character[LeftArrow]
839       then
840         begin
841           NewFuncTabPtr^.result := true { explicit result };
842           NewFuncTabPtr^.ResultName := name1;
843           position := position + 1;
844           identifier(name1, ItsAnIdentifier);
845           if not ItsAnIdentifier then
846             begin
847               SError(6)
848               { unrecognizable name to right of explicit res };
849               FuncHeadError := true
850             end
851           else NewFuncTabPtr^.result := false { no explicit result };
852           SkipSpaces;
853           if (position <= LineLength) and (not FuncHeadError)
854           then
855             begin
856               identifier(name2, ItsAnIdentifier);
857               if not ItsAnIdentifier
858               then
859                 begin
860                   SError(7) { invalid function/argument name };
861                   FuncHeadError := true
862                 end
863               else ArityIndex := 2
864               end;
865             SkipSpaces;
866             if (position <= LineLength) and (not FuncHeadError)
867             then
868               begin
869                 identifier(name3, ItsAnIdentifier);
870                 if not ItsAnIdentifier
871                 then
872                   begin
873                     SError(9) { invalid function right argument name };
874                     FuncHeadError := true
875                   end
876                 else ArityIndex := 3
877                 end;
878             SkipSpaces;
879

```

```

880 if (position <= LineLength) and (not FuncHeadError) then
881   begin
882     SError(3)
883     { extraneous characters to right of function header };
884     FuncHeadError := true
885   end;
886   case ArityIndex of
887   1:
888     begin
889       NewFuncTabPtr^.arity := niladic;
890       NewFuncTabPtr^.FuncName := name1;
891     end;
892   2:
893     begin
894       NewFuncTabPtr^.arity := monadic;
895       NewFuncTabPtr^.FuncName := name1;
896       NewFuncTabPtr^.RightArg := name2;
897       AddNameToVarTable(name2);
898       NewVarTabPtr^.FuncTabPtr := NewFuncTabPtr;
899     end;
900   3:
901     begin
902       NewFuncTabPtr^.arity := dyadic;
903       NewFuncTabPtr^.LeftArg := name1;
904       NewFuncTabPtr^.FuncName := name2;
905       NewFuncTabPtr^.RightArg := name3;
906       AddNameToVarTable(name1);
907       NewVarTabPtr^.FuncTabPtr := NewFuncTabPtr;
908       AddNameToVarTable(name3);
909       NewVarTabPtr^.FuncTabPtr := NewFuncTabPtr;
910     end
911   end { case };
912   if FunctionAlreadyDefined(NewFuncTabPtr^.FuncName, DummyPtr)
913   then
914     begin
915       SError(5) { function already defined };
916       FuncHeadError := true;
917     end;
918   if FuncHeadError then
919     begin
920       dispose(NewFuncTabPtr) { header no good };
921       FunctionMode := false { exit function mode };
922       NewFuncTabPtr := OldFuncTabPtr;
923     end
924   end { processfuncheader };
925
926
927
928 procedure DestroyStatement;
929
930 var
931   DumTokenPtr: ^TokenTable;
932   AuxSubrTabPtr: ^SubrTab;
933
934 begin
935   if SubrTabPtr <> nil
936   then
937     begin
938       while SubrTabPtr^.LastSubrPtr <> nil do
939         begin
940           AuxSubrTabPtr := SubrTabPtr;
941           SubrTabPtr := SubrTabPtr^.LastSubrPtr;
942           dispose(AuxSubrTabPtr);
943         end;
944       dispose(SubrTabPtr);
945     end;
946   DumTokenPtr := OldTokenPtr;
947   while DumTokenPtr <> HoldTokenPtr do
948     begin
949       OldTokenPtr := OldTokenPtr^.NextToken; dispose(DumTokenPtr);
950       DumTokenPtr := OldTokenPtr
951     end;
952   NewTokenPtr := HoldTokenPtr;
953   OldTokenPtr := HoldTokenPtr
954   { return pointer to end of last good line }
955 end { destroystatement };
956
957
958 procedure ReverseLinkList(var ArgPtr: TypeValTabPtr);
959
960 var
961   hold, TempPtr: ^values;
962
963 begin { reverselinklist }
964   ValPtr := ArgPtr^.FirstValue; TempPtr := ValPtr^.NextValue;
965   while TempPtr <> nil do
966     begin
967       hold := TempPtr^.NextValue; TempPtr^.NextValue := ValPtr;
968       ValPtr := TempPtr; TempPtr := hold
969     end;
970   ArgPtr^.FirstValue^.NextValue := nil;
971   ArgPtr^.FirstValue := ValPtr;
972   if ArgPtr^.ForwardOrder
973   then ArgPtr^.ForwardOrder := false
974   else ArgPtr^.ForwardOrder := true { toggle list order switch }
975 end { reverselinklist };
976
977
978 procedure parser(var TokenTabPtr: TokenPtr; var PtrToDa: TypeValTabPtr);
979
980 var

```

```

981 VFuncHold: "vfunc { hold while searching };
982 AuxOperTabPtr: "OperandTab;
983 AuxSubrTabPtr: "SubrTab;
984 AuxRParamPtr: "RParamTab;
985 AuxLParamPtr: "LParamTab;
986 ValidExp: Boolean { true if valid expression };
987 cnt: integer;
988 npv: integer { number of indices };
989 assign, assign1: Boolean { assign.in progress };
990 DoneSuccessor: Boolean;
991 DoneParse: Boolean;
992
993 procedure error(ErrorIndex: integer);
994
995 var
996   MsgCol: integer;
997
998 begin
999   write(' ', ErrorIndex, ' ');
1000   for MsgCol := 1 to MessageLength do
1001     write(ErrorMsgs[ErrorIndex, MsgCol]);
1002   writeln; goto 100 { return to scanner };
1003 end { error };
1004
1005
1006 procedure release;
1007
1008 begin { releaseopertab }
1009   OperTabPtr := PtrLastOper;
1010   while OperTabPtr.LastOper <> nil do
1011     begin
1012       AuxOperTabPtr := OperTabPtr;
1013       OperTabPtr := OperTabPtr.LastOper; dispose(AuxOperTabPtr);
1014     end;
1015   end { releaseopertab };
1016
1017
1018 procedure expression(var ValidExp: Boolean);
1019 forward;
1020
1021
1022 procedure ReturnToCallingSubr;
1023
1024 var
1025   NamePtr: "VarTab;
1026
1027 begin { returntocallingsubr }
1028   if SubrTabPtr.CalledSubr.result
1029   then
1030     begin { place explicit result in opertab }
1031       if not NameInVarTable(SubrTabPtr.CalledSubr.ResultName,
1032         NamePtr, SubrTabPtr.CalledSubr)
1033       then error(11) { 'symbol not found' }
1034       else
1035         begin
1036           AuxOperTabPtr := OperTabPtr; new(OperTabPtr);
1037           OperTabPtr.LastOper := AuxOperTabPtr;
1038           PtrLastOper := OperTabPtr;
1039           OperTabPtr.OperPtr := NamePtr.ValTabPtr;
1040         end;
1041       end;
1042     { return to calling function }
1043     VFuncPtr := SubrTabPtr.StateCallingSubr;
1044     TokenTabPtr := SubrTabPtr.TokenCallingSubr.NextToken;
1045     if SubrTabPtr.CalledSubr.arity <> niladic
1046     then
1047       begin { monadic or dyadic }
1048         AuxRParamPtr := RParamPtr; RParamPtr := RParamPtr.LastParam;
1049         dispose(AuxRParamPtr);
1050         if SubrTabPtr.CalledSubr.arity = dyadic then
1051           begin { dyadic only }
1052             AuxLParamPtr := LParamPtr;
1053             LParamPtr := LParamPtr.LastParam; dispose(AuxLParamPtr);
1054           end;
1055         end;
1056       AuxSubrTabPtr := SubrTabPtr;
1057       SubrTabPtr := SubrTabPtr.LastSubrPtr; dispose(AuxSubrTabPtr);
1058     end { returntocallingsubr };
1059
1060
1061 function SpecSymbol(sym: integer): Boolean;
1062
1063 var
1064   ValidSym: Boolean;
1065
1066 begin { specsymbol }
1067   ValidSym := false;
1068   if TokenTabPtr.noun = SpecOper
1069   then
1070     if TokenTabPtr.CharIndx = sym then
1071       begin
1072         hold := TokenTabPtr;
1073         TokenTabPtr := TokenTabPtr.NextToken; ValidSym := true;
1074       end;
1075     SpecSymbol := ValidSym;
1076   end { specsymbol };
1077
1078
1079 procedure CallSubr;
1080
1081

```

```

1082 var
1083   PtrToVarTab: "VarTab;
1084
1085 begin { callsubr }
1086   if SubrTabPtr.CalledSubr.arity <> niladic
1087   then
1088     begin
1089       if not NameInVarTable(SubrTabPtr.CalledSubr.RightArg,
1090         PtrToVarTab, SubrTabPtr.CalledSubr)
1091       then error(32);
1092       if PtrToVarTab.FuncTabPtr <> SubrTabPtr.CalledSubr
1093       then error(32) { program logic error, variable name of };
1094       { function argument not found in symbol table }
1095       AuxRParamPtr := RParamPtr; new(RParamPtr);
1096       RParamPtr.LastParam := AuxRParamPtr;
1097       PtrToVarTab.DeferredValTabPtr := RParamPtr;
1098       if SubrTabPtr.CalledSubr.arity = dyadic
1099       then
1100         begin { if dyadic }
1101           if not NameInVarTable(SubrTabPtr.CalledSubr.LeftArg,
1102             PtrToVarTab, SubrTabPtr.CalledSubr)
1103           then error(33);
1104           if PtrToVarTab.FuncTabPtr <> SubrTabPtr.CalledSubr
1105           then error(33) { same as error(32) };
1106           AuxLParamPtr := LParamPtr; new(LParamPtr);
1107           LParamPtr.LastParam := AuxLParamPtr;
1108           PtrToVarTab.DeferredValTabPtr := LParamPtr;
1109           LParamPtr.PtrVal := OperTabPtr.OperPtr;
1110           AuxOperTabPtr := OperTabPtr;
1111           OperTabPtr := OperTabPtr.LastOper;
1112           dispose(AuxOperTabPtr); PtrLastOper := OperTabPtr;
1113         end;
1114         RParamPtr.PtrVal := OperTabPtr.OperPtr;
1115         AuxOperTabPtr := OperTabPtr;
1116         OperTabPtr := OperTabPtr.LastOper; dispose(AuxOperTabPtr);
1117         PtrLastOper := OperTabPtr;
1118       end;
1119       TokenTabPtr := SubrTabPtr.CalledSubr.FirstStatement.NextStmnt;
1120       VFuncPtr := SubrTabPtr.CalledSubr.FirstStatement;
1121     end { callsubr };
1122
1123
1124 function FuncCall: Boolean;
1125
1126 var
1127   PtrToFuncTab: "FuncTab;
1128   NameOfFunc: PackedString;
1129   ValidFn: Boolean;
1130
1131 begin { funcall }
1132   ValidFn := false;
1133   if TokenTabPtr.noun = GlobVar
1134   then
1135     begin
1136       NameOfFunc := TokenTabPtr.VarTabPtr.VarName;
1137       if FunctionAlreadyDefined(NameOfFunc, PtrToFuncTab)
1138       then
1139         begin
1140           AuxSubrTabPtr := SubrTabPtr; new(SubrTabPtr);
1141           SubrTabPtr.LastSubrPtr := AuxSubrTabPtr;
1142           SubrTabPtr.CalledSubr := PtrToFuncTab;
1143           SubrTabPtr.TokenCallingSubr := TokenTabPtr;
1144           SubrTabPtr.StateCallingSubr := VFuncPtr;
1145           hold := TokenTabPtr;
1146           TokenTabPtr := TokenTabPtr.NextToken; ValidFn := true;
1147         end;
1148       end;
1149       FuncCall := ValidFn;
1150     end { funcall };
1151
1152
1153 procedure NumWrite(RealNo: real);
1154
1155 var
1156   prefix, root: integer;
1157   SigDig, ColCnt: integer;
1158
1159 begin { output a number }
1160   if RealNo >= 0.0
1161   then write(' ', RealNo: 12: 2) { output positive number }
1162   else
1163     begin { output negative number }
1164       RealNo := - 1.0 * RealNo;
1165       SigDig := trunc((ln(RealNo)) / (ln(10.0)));
1166       for ColCnt := 1 to (7 - SigDig) do write(' ');
1167       if character[negative] < 6000
1168       then write(chr(character[negative]));
1169     else
1170       begin
1171         prefix := character[negative] div 100;
1172         root := character[negative] - (100 * prefix);
1173         write(chr(prefix), chr(root));
1174       end;
1175       SigDig := SigDig + 5; write(RealNo: SigDig: 2);
1176     end { numwrite };
1177
1178
1179
1180 procedure OutPutVal;
1181

```

```

1182 var
1183   cnt: integer;
1184   AuxValuesPtr: ^values;
1185   DimHold, dimen1, dimen2, dimen3: integer;
1186   OutCnt1, OutCnt2, OutCnt3: integer;
1187   idimens: integer;
1188
1189 begin [ outputval ]
1190   cnt := 0; writeln; writeln;
1191   if not OperTabPtr^.OperPtr^.ForwardOrder
1192   then ReverseLinkList(OperTabPtr^.OperPtr);
1193   AuxValuesPtr := OperTabPtr^.OperPtr^.FirstValue;
1194   idimens := OperTabPtr^.OperPtr^.dimensions;
1195   if not (idimens in [0..3])
1196   then
1197     begin
1198       for ColCnt := 1 to MessageLength do
1199         write(ErrorMsgs[60], ColCnt);
1200       writeln;
1201     end
1202   else
1203     if AuxValuesPtr = nil
1204     then
1205       begin
1206         for ColCnt := 1 to MessageLength do
1207           write(ErrorMsgs[61], ColCnt);
1208         writeln;
1209       end
1210     else
1211       if idimens = 0
1212       then begin MunWrite(AuxValuesPtr^.RealVal); writeln; end
1213       else
1214         begin
1215           dimen1 := OperTabPtr^.OperPtr^.FirstDimen^.dimenlength;
1216           if idimens >= 2
1217           then
1218             dimen2 := OperTabPtr^.OperPtr^.FirstDimen^.NextDimen
1219               ^dimenlength
1220           else dimen2 := 1;
1221           if idimens = 3
1222           then
1223             dimen3 := OperTabPtr^.OperPtr^.FirstDimen^.NextDimen
1224               ^NextDimen^.dimenlength
1225           else dimen3 := 1;
1226           if idimens = 3 then
1227             begin [ rotate dimensions ]
1228               DimHold := dimen1; dimen1 := dimen2;
1229               dimen2 := dimen3; dimen3 := DimHold;
1230             end;
1231           for OutCnt3 := 1 to dimen3 do
1232             begin
1233               for OutCnt2 := 1 to dimen2 do
1234                 begin
1235                   for OutCnt1 := 1 to dimen1 do
1236                     begin
1237                       cnt := cnt + 1;
1238                       if ((cnt - 1) mod 5) = 0 and (cnt <> 1)
1239                       then begin writeln; write(' '); end;
1240                       MunWrite(AuxValuesPtr^.RealVal);
1241                       AuxValuesPtr := AuxValuesPtr^.NextValue;
1242                     end;
1243                   if idimens >= 2
1244                   then begin writeln; cnt := 0; end;
1245                 end;
1246               writeln; writeln;
1247             end;
1248           ( writeln; )
1249         end;
1250       end [ outputval ];
1251
1252 function variable: Boolean;
1253
1254 var
1255   globOrDummy: Boolean ( gord );
1256   PassedAdj: ^VarTab ( k );
1257   rarg: Boolean ( rd );
1258   ParmPtr: ^ValTab ( pt );
1259   ValidVar: Boolean;
1260   ValidIndex: Boolean;
1261
1262 procedure InputVal;
1263
1264 var
1265   AuxPtrToDa: ^ValTab;
1266   AuxValuesPtr: ^values;
1267   Aux2ValuesPtr: ^values;
1268   RealV: real;
1269   boolv: Boolean;
1270   cntnr, cnt: integer;
1271   AuxDimenFoPtr: ^DimenInfo;
1272
1273 begin [ inputval ]
1274   cnt := 0; position := 1; AuxPtrToDa := PtrToDa;
1275   new(PtrToDa); AuxPtrToDa^.NextValTabLink := PtrToDa;
1276   AuxOperTabPtr := OperTabPtr; new(OperTabPtr);
1277   PtrLastOper := OperTabPtr;
1278   OperTabPtr^.LastOper := AuxOperTabPtr;
1279   OperTabPtr^.OperPtr := PtrToDa; new(Aux2ValuesPtr);
1280   PtrToDa^.FirstValue := Aux2ValuesPtr;

```

```

1281   for cntnr := 1 to MessageLength do write(ErrorMsgs[63], cntnr);
1282   writeln; readln; GetAPLstatement;
1283   repeat
1284     MakeNumber(RealV, boolv); SkipSpaces;
1285     if not boolv
1286     then
1287       begin
1288         for ColCnt := 1 to MessageLength do
1289           write(ErrorMsgs[62], ColCnt);
1290         writeln; position := 1; cnt := 0;
1291         Aux2ValuesPtr := OperTabPtr^.OperPtr^.FirstValue;
1292         for cntnr := 1 to MessageLength do
1293           write(ErrorMsgs[63], cntnr);
1294         writeln; readln; GetAPLstatement
1295       end
1296     else
1297       begin
1298         cnt := cnt + 1; AuxValuesPtr := Aux2ValuesPtr;
1299         new(Aux2ValuesPtr); AuxValuesPtr^.RealVal := RealV;
1300         AuxValuesPtr^.NextValue := Aux2ValuesPtr;
1301       end;
1302     until position > LineLength;
1303     dispose(Aux2ValuesPtr); AuxValuesPtr^.NextValue := nil;
1304     PtrToDa^.IntermedResult := false; PtrToDa^.dimensions := 1;
1305     PtrToDa^.ForwardOrder := true;
1306     PtrToDa^.NextValTabLink := nil; new(AuxDimenFoPtr);
1307     PtrToDa^.FirstDimen := AuxDimenFoPtr;
1308     AuxDimenFoPtr^.dimenlength := cnt;
1309     AuxDimenFoPtr^.NextDimen := nil;
1310   end [ inputval ];
1311
1312 procedure GetArrayPosition(var ValuesPtr: TypeValuesPtr);
1313
1314 var
1315   indice: real;
1316   kcnc: integer;
1317   sl: integer;
1318   AuxDimenFoPtr: ^DimenInfo;
1319
1320 begin [ getarrayposition ]
1321   if npv <> ParmPtr^.dimensions then error(35);
1322   [ 'wrong num. of subscripts' ]
1323   sl := 0; AuxOperTabPtr := OperTabPtr;
1324   AuxDimenFoPtr := ParmPtr^.FirstDimen;
1325   for kcnc := 1 to npv do
1326     begin
1327       if AuxOperTabPtr^.OperPtr^.dimensions <> 0
1328       then error(35) [ 'non-scalar indices' ];
1329       indice := AuxOperTabPtr^.OperPtr^.FirstValue^.RealVal;
1330       if indice - 1.0 * trunc(indice) <> 0.0
1331       then error(37) [ 'non-integer indices' ];
1332       if not (trunc(indice) in [1..AuxDimenFoPtr^.dimenlength
1333         ])
1334       then error(38) [ 'out of range index' ];
1335       sl := (sl * AuxDimenFoPtr^.dimenlength) + trunc(indice) -
1336         1;
1337       AuxOperTabPtr := AuxOperTabPtr^.LastOper;
1338       dispose(OperTabPtr); OperTabPtr := AuxOperTabPtr;
1339       AuxDimenFoPtr := AuxDimenFoPtr^.NextDimen;
1340     end;
1341   ValuesPtr := ParmPtr^.FirstValue;
1342   while sl <> 0 do [ determine which value in ]
1343     [ pt[sval(sv)][sval(sv-1)]...[sval(sv-npv+1)] ]
1344     [ := sval(sv-npv) ]
1345     begin ValuesPtr := ValuesPtr^.NextValue; sl := sl - 1; end;
1346   end [ getarrayposition ];
1347
1348 procedure LinkResults;
1349
1350 var
1351   PtrToValues: ^values;
1352
1353 begin [ linkresults ]
1354   if npv = 0
1355   then
1356     begin
1357       if not globOrDummy
1358       then
1359         if rarg then RParmPtr^.PtrVal := OperTabPtr^.OperPtr
1360         else LParmPtr^.PtrVal := OperTabPtr^.OperPtr
1361         else PassedAdj^.ValTabPtr := OperTabPtr^.OperPtr
1362       end
1363     else
1364       begin
1365         if globOrDummy then ParmPtr := PassedAdj^.ValTabPtr
1366         else ParmPtr := PassedAdj^.DeferredValTabPtr^.PtrVal;
1367         GetArrayPosition(PtrToValues);
1368         if OperTabPtr^.OperPtr^.dimensions <> 0
1369         then error(36) [ 'assigned expression not a scalar' ];
1370         PtrToValues^.RealVal := OperTabPtr^.OperPtr^.FirstValue
1371           ^RealVal;
1372       end;
1373     AuxOperTabPtr := OperTabPtr;
1374     OperTabPtr := OperTabPtr^.LastOper; dispose(AuxOperTabPtr);
1375     PtrLastOper := OperTabPtr;
1376   end [ linkresults ];
1377
1378 procedure StackPointers;

```

```

1384 var
1385   AuxPtrToDa: ValTab;
1386   PtrToValues, AuxValuesPtr: values;
1387
1388 begin ( stackpointers )
1389   ;
1390   if npv = 0
1391   then
1392     begin
1393       AuxOperTabPtr := OperTabPtr; new(OperTabPtr);
1394       OperTabPtr.LastOper := AuxOperTabPtr;
1395       OperTabPtr.OperPtr := ParmPtr;
1396       PtrLastOper := OperTabPtr
1397     end
1398   else
1399     begin
1400       AuxPtrToDa := PtrToDa; new(PtrToDa);
1401       PtrToDa.NextValTabLink := AuxPtrToDa;
1402       PtrToDa.IntermedResult := true;
1403       PtrToDa.dimensions := 0; PtrToDa.FirstDimen := nil;
1404       PtrToDa.ForwardOrder := true; new(AuxValuesPtr);
1405       PtrToDa.FirstValue := AuxValuesPtr;
1406       GetArrayPosition(PtrToValues);
1407       PtrToDa.FirstValue.RealVal := PtrToValues.RealVal;
1408       PtrToDa.FirstValue.NextValue := nil;
1409       AuxOperTabPtr := OperTabPtr; new(OperTabPtr);
1410       OperTabPtr.LastOper := AuxOperTabPtr;
1411       OperTabPtr.OperPtr := PtrToDa;
1412       PtrLastOper := OperTabPtr;
1413     end;
1414   end ( stackpointers );
1415
1416 function SimpleVariable: Boolean;
1417
1418 var
1419   ValidSv: Boolean;
1420
1421 begin ( simplevariable )
1422   ValidSv := false; rarg := false; globOrDummy := false;
1423   if assign
1424   then
1425     begin
1426       if (TokenTabPtr.noun = FormRes) or (TokenTabPtr.noun =
1427         GlobVar)
1428       then
1429         begin
1430           globOrDummy := true;
1431           PassedAdj := TokenTabPtr.VarTabPtr;
1432           hold := TokenTabPtr;
1433           TokenTabPtr := TokenTabPtr.NextToken;
1434           ValidSv := true
1435         end
1436       else
1437         if TokenTabPtr.noun = FormArg
1438         then
1439           begin
1440             if NamesMatch(TokenTabPtr.VarTabPtr.FuncTabPtr.
1441               LeftArg, TokenTabPtr.VarTabPtr.VarName)
1442             then rarg := true;
1443             PassedAdj := TokenTabPtr.VarTabPtr
1444           end
1445         end
1446       end
1447     else
1448       begin
1449         if (TokenTabPtr.noun = FormRes) or (TokenTabPtr.noun =
1450           GlobVar)
1451         then
1452           begin
1453             ParmPtr := TokenTabPtr.VarTabPtr.ValTabPtr;
1454             if ParmPtr <> nil then
1455               begin
1456                 hold := TokenTabPtr;
1457                 TokenTabPtr := TokenTabPtr.NextToken;
1458                 ValidSv := true
1459               end
1460             end
1461           else
1462             begin
1463               if TokenTabPtr.noun = FormArg
1464               then
1465                 begin
1466                   if NamesMatch(TokenTabPtr.VarTabPtr.FuncTabPtr.
1467                     LeftArg, TokenTabPtr.VarTabPtr.VarName)
1468                   then ParmPtr := LParmPtr.PtrVal;
1469                   else ParmPtr := RParmPtr.PtrVal;
1470                   hold := TokenTabPtr;
1471                   TokenTabPtr := TokenTabPtr.NextToken;
1472                   ValidSv := true;
1473                 end;
1474               end;
1475             end;
1476           SimpleVariable := ValidSv;
1477         end ( simple variable );
1478
1479 procedure index(var ValidI: Boolean);
1480
1481 var
1482   ValidE1, ValidE2: Boolean;
1483
1484   begin ( index )
1485     ValidI := false; expression(ValidE1);
1486     if ValidE1
1487     then
1488       begin
1489         npv := 1 ( no. of index expressions );
1490         while SpecSymbol(XSemicolon) do
1491           begin
1492             npv := npv + 1; expression(ValidE2);
1493             if not ValidE2 then error(39);
1494             ( 'invalid index expression' )
1495           end;
1496           ValidI := true;
1497         end;
1498       end ( index );
1499
1500   begin ( variable )
1501     ValidVar := false; npv := 0;
1502     if not assign
1503     then
1504       if SpecSymbol(XQuadSym)
1505       then begin InputVal; ValidVar := true end
1506       else
1507         begin
1508           if SpecSymbol(XRightBracket)
1509           then
1510             begin
1511               index(ValidIndex);
1512               if (not ValidIndex) or (not SpecSymbol(XLeftBracket))
1513               then error(34) ( 'invalid index expression' );
1514             end;
1515             if SimpleVariable
1516             then begin StackPointers; ValidVar := true end
1517           end
1518         else
1519           if SpecSymbol(XQuadSym)
1520           then begin OutPutVal; ValidVar := true end
1521           else
1522             begin
1523               if SpecSymbol(XRightBracket)
1524               then
1525                 begin
1526                   index(ValidIndex);
1527                   if (not ValidIndex) or (not SpecSymbol(XLeftBracket))
1528                   then error(34) ( 'invalid index expression' );
1529                 end;
1530                 if SimpleVariable
1531                 then begin LinkResults; ValidVar := true; end;
1532               end;
1533             end;
1534             variable := ValidVar;
1535           end ( variable );
1536
1537   procedure primary(var valid: Boolean) ( recursive entry );
1538   var
1539     ValidX: Boolean;
1540     assign: Boolean;
1541
1542   function vector: Boolean;
1543
1544   var
1545     vec: Boolean;
1546
1547   begin ( vector )
1548     vec := false;
1549     if TokenTabPtr.noun = constant
1550     then
1551       begin
1552         AuxOperTabPtr := OperTabPtr; new(OperTabPtr);
1553         PtrLastOper := OperTabPtr;
1554         OperTabPtr.LastOper := AuxOperTabPtr;
1555         OperTabPtr.OperPtr := TokenTabPtr.ValTabPtr;
1556         hold := TokenTabPtr;
1557         TokenTabPtr := TokenTabPtr.NextToken; vec := true;
1558       end;
1559     vector := vec;
1560   end ( vector );
1561
1562   begin ( primary )
1563     valid := true;
1564     if not vector
1565     then
1566       begin
1567         assign := false;
1568         if not variable
1569         then
1570           if SpecSymbol(XRightPar)
1571           then
1572             begin
1573               expression(ValidX);
1574               if not ValidX
1575               then error(14) ( 'non-valid exp within parens' );
1576             end;
1577           else
1578             if not SpecSymbol(XLeftPar)
1579             then
1580               error(15)
1581               ( 'right paren not balanced with left paren' );
1582             else valid := true
1583           end
1584         end
1585       end
1586     end
1587   end

```

```

1588     end
1589     else
1590     if not FunctCall then valid := false
1591     else begin CallSubr; primary(valid); end;
1592 end;
1593 end ( primary );
1594
1595 procedure expression ( recursive );
1596
1597 var
1598 DoneExp, ValidPri, ValidFunc, ValidAssn: Boolean;
1599 code: integer;
1600
1601
1602
1603 procedure assignment(var valida: Boolean);
1604
1605 begin ( assignment )
1606 valida := false;
1607 if SpecSymbol(XLeftArrow)
1608 then
1609 begin
1610 assign := true; assign1 := true;
1611 if variable then valida := true
1612 else error(8) ( result of an assn not a valid variable );
1613 valida := true; assign := false;
1614 end;
1615 end ( assignment );
1616
1617
1618 function mop: Boolean;
1619
1620 var
1621 ValidM: Boolean;
1622
1623 begin ( mop )
1624 ValidM := false;
1625 if (TokenTabPtr^.noun = MonadOper) or (TokenTabPtr^.noun =
1626 ReductOper)
1627 then
1628 begin
1629 if TokenTabPtr^.noun = MonadOper
1630 then code := MOpTab[TokenTabPtr^.MonIndex].OpIndex
1631 else code := RedTab[TokenTabPtr^.RedIndex].OpIndex;
1632 hold := TokenTabPtr;
1633 TokenTabPtr := TokenTabPtr^.NextToken; ValidM := true;
1634 end;
1635 mop := ValidM;
1636 end ( mop );
1637
1638
1639 function dop: Boolean;
1640
1641 var
1642 ValidD: Boolean;
1643
1644 begin ( dop )
1645 ValidD := false;
1646 if TokenTabPtr^.noun = DyadOper
1647 then
1648 begin
1649 code := DOpTab[TokenTabPtr^.DOpIndx].OpIndex;
1650 hold := TokenTabPtr;
1651 TokenTabPtr := TokenTabPtr^.NextToken;
1652 if (code > 80) then ValidD := true
1653 else
1654 if TokenTabPtr^.noun = SpecOper
1655 then
1656 if SpecSymbol(XPeriod)
1657 then
1658 begin
1659 if TokenTabPtr^.noun = DyadOper
1660 then
1661 begin
1662 if DOpTab[TokenTabPtr^.DOpIndx].OpIndex <= 80
1663 then
1664 begin
1665 code := code + (100 * DOpTab[TokenTabPtr^.
1666 DOpIndx].OpIndex);
1667 hold := TokenTabPtr;
1668 TokenTabPtr := TokenTabPtr^.NextToken;
1669 ValidD := true
1670 end
1671 else error(27) ( 'invalid inner product exp' )
1672 end
1673 else
1674 if TokenTabPtr^.noun = SpecOper
1675 then
1676 begin
1677 if SpecSymbol(XLittleCircle)
1678 then
1679 begin code := 10 * code; ValidD := true
1680 end
1681 else error(26) ( 'inval outer prod exp' )
1682 end
1683 else error(26) ( same as above )
1684 end
1685 else ValidD := true
1686 else ValidD := true;
1687 end;
1688 dop := ValidD;

```

```

1689 end ( dop );
1690
1691 function ItsBoolean(test: real): Boolean;
1692
1693 begin
1694 if (test = 1.0) or (test = 0.0) then ItsBoolean := true
1695 else ItsBoolean := false
1696 end ( itsboolean );
1697
1698
1699 procedure DyadComp(var SFloat: real; value: real; code: integer);
1700 ( compute result of dyadic operation )
1701
1702 begin
1703 case code of
1704 ( left codes - reduction ops / right codes - dyadic ops )
1705 2, 52: SFloat := value + SFloat ( addition );
1706 3, 53: SFloat := value - SFloat ( subtraction );
1707 4, 54: SFloat := value * SFloat ( multiplication );
1708 5, 55:
1709 if SFloat = 0.0
1710 then error(20) ( attempted division by zero )
1711 else SFloat := value / SFloat ( division );
1712 6, 56:
1713 if value > 0.0
1714 then
1715 SFloat := exp(SFloat * ln(value))
1716 ( number raised to a power )
1717 else
1718 SFloat := 1.0 / (exp(SFloat * ln(abs(value))));
1719 21, 71:
1720 if value = SFloat ( equality ) then SFloat := 1.0
1721 else SFloat := 0.0;
1722 22, 72:
1723 if value <> SFloat ( inequality ) then SFloat := 1.0
1724 else SFloat := 0.0;
1725 23, 73:
1726 if value < SFloat ( less than ) then SFloat := 1.0
1727 else SFloat := 0.0;
1728 24, 74:
1729 if value <= SFloat ( less than or equal to )
1730 then SFloat := 1.0
1731 else SFloat := 0.0;
1732 25, 75:
1733 if value >= SFloat ( greater than or equal to )
1734 then SFloat := 1.0
1735 else SFloat := 0.0;
1736 26, 76:
1737 if value > SFloat ( greater than ) then SFloat := 1.0
1738 else SFloat := 0.0;
1739 27, 77:
1740 if (ItsBoolean(value)) and (ItsBoolean(SFloat))
1741 then
1742 if (value = 1.0) and (SFloat = 1.0) ( and )
1743 then SFloat := 1.0
1744 else SFloat := 0.0
1745 else error(19) ( value not boolean );
1746 28, 78:
1747 if (ItsBoolean(value)) and (ItsBoolean(SFloat))
1748 then
1749 if (value = 1.0) or (SFloat = 1.0) ( or )
1750 then SFloat := 1.0
1751 else SFloat := 0.0
1752 else error(19) ( value not boolean );
1753 29:
1754 if value > SFloat ( maximum or ceiling )
1755 then SFloat := value;
1756 30:
1757 if value < SFloat ( minimum or floor )
1758 then SFloat := value;
1759 31:
1760 if (value * SFloat) < 0.0
1761 then error(50) ( number and base of different sign )
1762 else
1763 SFloat := (ln(abs(SFloat))) / (ln(abs(value)))
1764 ( log to a base )
1765 end ( case )
1766 end ( dyadcomp );
1767
1768
1769 procedure IndexGenerator(arg: TypeValTabPtr);
1770 ( monadic iota operator )
1771
1772 var
1773 TotalIndex, TopValue: integer;
1774
1775 begin
1776 if arg^.dimensions <> 0
1777 then error(21) ( argument not a scalar )
1778 else
1779 if arg^.FirstValue^.RealVal < 0.0
1780 then error(22) ( argument is negative )
1781 else
1782 if (arg^.FirstValue^.RealVal) - (1.0 * trunc(arg^.
1783 FirstValue^.RealVal)) <> 0.0
1784 then error(23) ( argument is not an integer )
1785 else
1786 begin
1787 new(NewValTabLink);
1788 OldValTabLink^.NextValTabLink := NewValTabLink;

```

```

1789     NewValTabLink".NextValTabLink := nil;
1790     NewValTabLink".ForwardOrder := true;
1791     NewValTabLink".IntermedResult := true;
1792     NewValTabLink".dimensions := 1 ( result is a vector );
1793     new(NewDim);   NewValTabLink".FirstDimen := NewDim;
1794     TopValue := trunc(arg".FirstValue".RealVal)
1795     ( last index generd );
1796     NewDim".dimenlength := TopValue;
1797     NewDim".NextDimen := nil;   iotaIndex := 1;
1798     switch := true;
1799     while iotaIndex <= TopValue do
1800     begin
1801         new(NewValues);   NewValues".RealVal := iotaIndex;
1802         if switch
1803         then
1804             begin
1805                 switch := false;
1806                 NewValTabLink".FirstValue := NewValues
1807             end
1808             else NewValPtr".NextValue := NewValues;
1809                 NewValPtr := NewValues;
1810                 iotaIndex := iotaIndex + 1
1811             end;
1812         if switch
1813         then
1814             NewValTabLink".FirstValue := nil
1815             ( result is vector of length 0 )
1816             else NewValues".NextValue := nil
1817         end
1818     end ( indexgenerator );
1819
1820 procedure ravel(arg: TypeValTabPtr);
1821 { monadic comma operator }
1822
1823 var
1824     elements: integer;
1825
1826 begin
1827     new(NewValTabLink);
1828     OldValTabLink".NextValTabLink := NewValTabLink;
1829     NewValTabLink".NextValTabLink := nil;
1830     NewValTabLink".IntermedResult := true;
1831     NewValTabLink".ForwardOrder := arg".ForwardOrder;
1832     NewValTabLink".dimensions := 1 ( result is a vector );
1833     new(NewDim);   NewValTabLink".FirstDimen := NewDim;
1834     NewDim".NextDimen := nil;   switch := true;
1835     ValPtr := arg".FirstValue;   elements := 0;
1836     while ValPtr <> nil do
1837     begin ( duplicate values into result )
1838         new(NewValues);   NewValues".RealVal := ValPtr".RealVal;
1839         elements := elements + 1;
1840         if switch
1841         then
1842             begin
1843                 switch := false;
1844                 NewValTabLink".FirstValue := NewValues
1845             end
1846             else NewValPtr".NextValue := NewValues;
1847                 NewValPtr := NewValues;   ValPtr := ValPtr".NextValue
1848             end;
1849         NewDim".dimenlength := elements;
1850         if switch then NewValTabLink".FirstValue := nil
1851         else NewValues".NextValue := nil
1852     end ( ravel );
1853
1854 procedure ShapeOf(arg: TypeValTabPtr);
1855 { monadic rho operator }
1856
1857 begin
1858     new(NewValTabLink);
1859     OldValTabLink".NextValTabLink := NewValTabLink;
1860     NewValTabLink".NextValTabLink := nil;
1861     NewValTabLink".IntermedResult := true;
1862     NewValTabLink".ForwardOrder := true;
1863     NewValTabLink".dimensions := 1 ( result is a vector );
1864     new(NewDim);   NewDim".dimenlength := arg".dimensions;
1865     NewValTabLink".FirstDimen := NewDim;
1866     NewDim".NextDimen := nil;   switch := true;
1867     DimPtr := arg".FirstDimen;
1868     while DimPtr <> nil do
1869     begin ( argument dimensions become result values )
1870         new(NewValues);
1871         NewValues".RealVal := DimPtr".dimenlength;
1872         if switch
1873         then
1874             begin
1875                 switch := false;
1876                 NewValTabLink".FirstValue := NewValues
1877             end
1878             else NewValPtr".NextValue := NewValues;
1879                 NewValPtr := NewValues;   DimPtr := DimPtr".NextDimen
1880             end;
1881         if switch
1882         then
1883             NewValTabLink".FirstValue := nil
1884             ( result is a vector of length 0 )
1885         else NewValues".NextValue := nil
1886     end ( shapeof );
1887
1888

```

```

1891 procedure reduction(arg: TypeValTabPtr);
1892
1893 var
1894     counter, RowLength: integer;
1895     SFloat: real;
1896
1897 begin
1898     if (arg".dimensions = 0) or (arg".FirstValue = nil)
1899     then
1900         error(24) ( argument is a scalar or vector of length zero )
1901     else
1902         if (arg".dimensions = 1) and (arg".FirstDimen".dimenlength
1903         = 1)
1904         then error(51) ( argument is a vector of length one )
1905         else
1906             begin
1907                 new(NewValTabLink);
1908                 OldValTabLink".NextValTabLink := NewValTabLink;
1909                 NewValTabLink".NextValTabLink := nil;
1910                 NewValTabLink".IntermedResult := true;
1911                 if arg".ForwardOrder then ReverseLinkList(arg);
1912                 NewValTabLink".ForwardOrder := false;
1913                 NewValTabLink".dimensions := arg".dimensions - 1;
1914                 DimPtr := arg".FirstDimen;   switch := true;
1915                 while DimPtr".NextDimen <> nil do
1916                 begin ( build dimensions of result )
1917                     new(NewDim);
1918                     if switch
1919                     then
1920                         begin
1921                             switch := false;
1922                             NewValTabLink".FirstDimen := NewDim
1923                         end
1924                         else NewPtr".NextDimen := NewDim;
1925                             NewDim".dimenlength := DimPtr".dimenlength;
1926                             NewPtr := NewDim;   DimPtr := DimPtr".NextDimen
1927                         end;
1928                     if switch
1929                     then
1930                         NewValTabLink".FirstDimen := nil
1931                         ( arg is vector, result is scalar )
1932                     else NewDim".NextDimen := nil;
1933                         RowLength := DimPtr".dimenlength;
1934                         ValPtr := arg".FirstValue;   switch := true;
1935                         while ValPtr <> nil do
1936                         begin ( perform reduction )
1937                             SFloat := ValPtr".RealVal
1938                             ( sfloat gets last value in row );
1939                             ValPtr := ValPtr".NextValue;
1940                             for counter := 2 to RowLength do
1941                             begin
1942                                 DyadComp(SFloat, ValPtr".RealVal, code);
1943                                 ValPtr := ValPtr".NextValue
1944                             end;
1945                             new(NewValues);   NewValues".RealVal := SFloat;
1946                             if switch
1947                             then
1948                                 begin
1949                                     switch := false;
1950                                     NewValTabLink".FirstValue := NewValues
1951                                 end
1952                                 else NewValPtr".NextValue := NewValues;
1953                                     NewValPtr := NewValues
1954                                 end;
1955                             NewValues".NextValue := nil
1956                             end;
1957                         end ( reduction );
1958
1959 procedure monadic(arg: TypeValTabPtr; token: TokenPtr);
1960 { operations with codes between 1 and 31 }
1961
1962 begin
1963     if token".noun = ReductOper then reduction(arg)
1964     else
1965         if code > 20
1966         then
1967             case code of
1968                 21: IndexGenerator(arg);
1969                 22: ShapeOf(arg);
1970                 23: ravel(arg)
1971             end ( case )
1972         else
1973             begin
1974                 new(NewValTabLink);
1975                 OldValTabLink".NextValTabLink := NewValTabLink;
1976                 NewValTabLink".NextValTabLink := nil;
1977                 NewValTabLink".IntermedResult := true;
1978                 NewValTabLink".ForwardOrder := arg".ForwardOrder;
1979                 NewValTabLink".dimensions := arg".dimensions;
1980                 switch := true;   DimPtr := arg".FirstDimen;
1981                 while DimPtr <> nil do
1982                 begin ( duplicate dimensions of arg into result )
1983                     new(NewDim);
1984                     NewDim".dimenlength := DimPtr".dimenlength;
1985                     if switch
1986                     then
1987                         begin
1988                             switch := false;
1989                             NewValTabLink".FirstDimen := NewDim
1990                         end
1991

```

```

1992     else NewPtr".NextDimen := NewDim;
1993     NewPtr := NewDim; DimPtr := DimPtr".NextDimen
1994   end;
1995   if switch
1996   then
1997     NewValTabLink".FirstDimen := nil ( result is a scalar )
1998   else NewDim".NextDimen := nil;
1999   switch := true; ValPtr := arg".FirstValue;
2000   while ValPtr <> nil do
2001     begin
2002       new(NewValues);
2003       if switch = true
2004       then
2005         begin
2006           switch := false;
2007           NewValTabLink".FirstValue := NewValues
2008         end
2009       else NewValPtr".NextValue := NewValues;
2010       NewValPtr := NewValues;
2011       case code of
2012       1:
2013         if ItsBoolean(ValPtr".RealVal)
2014         ( logical negation )
2015         then
2016           NewValues".RealVal := 1.0 - ValPtr".RealVal
2017         else error(19) ( value not boolean );
2018       2:
2019         NewValues".RealVal := ValPtr".RealVal
2020         ( no-op );
2021       3:
2022         NewValues".RealVal := 0.0 - ValPtr".RealVal
2023         ( negation );
2024       4:
2025         if ValPtr".RealVal > 0.0 ( signum )
2026         then NewValues".RealVal := 1.0
2027         else
2028           if ValPtr".RealVal < 0.0
2029           then NewValues".RealVal := - 1.0;
2030       5:
2031         if ValPtr".RealVal = 0.0 ( reciprocal )
2032         then error(54) ( attempted inverse of zero )
2033         else
2034           NewValues".RealVal := 1.0 / ValPtr".RealVal;
2035       6: NewValues".RealVal := exp(ValPtr".RealVal)
2036       end ( case );
2037       ValPtr := ValPtr".NextValue
2038     end;
2039     if switch then NewValTabLink".FirstValue := nil
2040     else NewValues".NextValue := nil
2041   end
2042 end ( monadic );
2043
2044 procedure catenate(LeftArg, RightArg: TypeValTabPtr);
2045 ( dyadic comma operator - joins 2 arguments )
2046
2047 var
2048   ResultLength: integer;
2049
2050 begin ( catenate )
2051   if (RightArg".dimensions > 1) or (LeftArg".dimensions > 1)
2052   then error(53) ( argument(s) with rank greater than 1 )
2053   else
2054     begin
2055       new(NewValTabLink);
2056       OldValTabLink".NextValTabLink := NewValTabLink;
2057       NewValTabLink".NextValTabLink := nil;
2058       NewValTabLink".IntermedResult := true;
2059       if not LeftArg".ForwardOrder
2060       then ReverseLinkList(LeftArg);
2061       if not RightArg".ForwardOrder
2062       then ReverseLinkList(RightArg);
2063       NewValTabLink".ForwardOrder := true;
2064       NewValTabLink".dimensions := 1 ( result is a vector );
2065       new(NewDim); NewValTabLink".FirstDimen := NewDim;
2066       NewDim".NextDimen := nil; ResultLength := 0;
2067       if LeftArg".dimensions = 0
2068       then
2069         ResultLength := ResultLength + 1 ( left arg is a scalar )
2070       else
2071         ResultLength := ResultLength + LeftArg".FirstDimen".
2072         dimenlength;
2073       if RightArg".dimensions = 0
2074       then
2075         ResultLength := ResultLength + 1 ( right arg is a scalar )
2076       else
2077         ResultLength := ResultLength + RightArg".FirstDimen".
2078         dimenlength;
2079       NewDim".dimenlength := ResultLength; switch := true;
2080       if ResultLength = 0
2081       then
2082         NewValTabLink".FirstValue := nil
2083         ( result is vector of length 0 )
2084       else
2085         begin ( transfer values to result )
2086           LeftValPtr := LeftArg".FirstValue;
2087           while LeftValPtr <> nil do
2088             begin ( transfer left arg values (if any) )
2089               new(NewValues);
2090               if switch
2091               then

```

```

2092         begin
2093           switch := false;
2094           NewValTabLink".FirstValue := NewValues
2095         end
2096       else NewValPtr".NextValue := NewValues;
2097       NewValues".RealVal := LeftValPtr".RealVal;
2098       NewValPtr := NewValues;
2099       LeftValPtr := LeftValPtr".NextValue
2100     end;
2101     RightValPtr := RightArg".FirstValue;
2102     while RightValPtr <> nil do
2103       begin ( transfer right arg values (if any) )
2104         new(NewValues);
2105         if switch
2106         then
2107           begin
2108             switch := false;
2109             NewValTabLink".FirstValue := NewValues
2110           end
2111         else NewValPtr".NextValue := NewValues;
2112         NewValues".RealVal := RightValPtr".RealVal;
2113         NewValPtr := NewValues;
2114         RightValPtr := RightValPtr".NextValue
2115       end;
2116       NewValues".NextValue := nil
2117     end ( transfer of values )
2118   end
2119 end ( catenate );
2120
2121 procedure IndexOf(LeftArg, RightArg: TypeValTabPtr);
2122 ( dyadic iota operator )
2123
2124 var
2125   MapIndex, icount, TestLength, OneMore: integer;
2126
2127 begin ( indexof )
2128   if LeftArg".dimensions <> 1
2129   then error(29) ( left argument is not a vector )
2130   else
2131     begin
2132       new(NewValTabLink);
2133       OldValTabLink".NextValTabLink := NewValTabLink;
2134       NewValTabLink".NextValTabLink := nil;
2135       NewValTabLink".IntermedResult := true;
2136       if not LeftArg".ForwardOrder
2137       then ReverseLinkList(LeftArg);
2138       NewValTabLink".ForwardOrder := RightArg".ForwardOrder;
2139       NewValTabLink".dimensions := RightArg".dimensions;
2140       if RightArg".dimensions = 0
2141       then
2142         NewValTabLink".FirstDimen := nil
2143         ( right argument is a scalar )
2144       else
2145         begin ( build dimensions of result )
2146           switch := true; DimPtr := RightArg".FirstDimen;
2147           while DimPtr <> nil do
2148             begin
2149               new(NewDim);
2150               if switch
2151               then
2152                 begin
2153                   switch := false;
2154                   NewValTabLink".FirstDimen := NewDim
2155                 end
2156               else NewPtr".NextDimen := NewDim;
2157               NewDim".dimenlength := DimPtr".dimenlength;
2158               NewPtr := NewDim; DimPtr := DimPtr".NextDimen
2159             end;
2160             NewDim".NextDimen := nil
2161           end;
2162           switch := true; RightValPtr := RightArg".FirstValue;
2163           while RightValPtr <> nil do
2164             begin
2165               new(NewValues);
2166               if switch
2167               then
2168                 begin
2169                   switch := false;
2170                   NewValTabLink".FirstValue := NewValues
2171                 end
2172               else NewValPtr".NextValue := NewValues;
2173               icount := 1; LeftValPtr := LeftArg".FirstValue;
2174               TestLength := LeftArg".FirstDimen".dimenlength
2175               ( length of left arg );
2176               OneMore := TestLength + 1
2177               ( length of left arg plus one );
2178               MapIndex := OneMore;
2179               while (icount <= TestLength) and (MapIndex = OneMore) do
2180                 begin
2181                   ( try to match value in right arg with one in left arg )
2182                   if LeftValPtr".RealVal = RightValPtr".RealVal
2183                   then MapIndex := icount ( value match );
2184                   icount := icount + 1;
2185                   LeftValPtr := LeftValPtr".NextValue
2186                 end;
2187               NewValues".RealVal := MapIndex;
2188               NewValPtr := NewValues;
2189               RightValPtr := RightValPtr".NextValue
2190             end
2191           end
2192         ( if no match, index becomes one more than length of left arg )
2193       ;
2194

```

```

2195     NewValues".NextValue := nil
2196   end
2197 end ( indexof );
198
199
2200 procedure reshape(LeftArg, RightArg: TypeValTabPtr);
2201 ( dyadic rho operator - change dimensions of )
2202
2203 var
2204   ResultLength, elements: integer;
2205   DimPtr: "DimenInfo;
2206   NewPtr: "values;
2207
2208 begin ( reshape )
2209   if LeftArg".dimensions > 1
2210   then error(50) ( left argument not a vector or a scalar )
2211   else
2212     begin
2213       new(NewValTabLink);
2214       OldValTabLink".NextValTabLink := NewValTabLink;
2215       NewValTabLink".NextValTabLink := nil;
2216       NewValTabLink".IntermedResult := true;
2217       if not LeftArg".ForwardOrder
2218       then ReverseLinkList(LeftArg);
2219       if not RightArg".ForwardOrder
2220       then ReverseLinkList(RightArg);
2221       NewValTabLink".ForwardOrder := true;
2222       if LeftArg".FirstDimen = nil
2223       then NewValTabLink".dimensions := 1
2224       else
2225         NewValTabLink".dimensions := LeftArg".FirstDimen".
2226         dimenlength;
2227       ResultLength := 1; LeftValPtr := LeftArg".FirstValue;
2228       switch := true;
2229       while LeftValPtr <> nil do
2230         ( left arg values are dimensions of result )
2231         begin ( build result dimensions )
2232           ResultLength := ResultLength * trunc(LeftValPtr".
2233             RealVal);
2234           new(NewDim);
2235           NewDim".dimenlength := trunc(LeftValPtr".RealVal);
2236           LeftValPtr := LeftValPtr".NextValue;
2237           if switch
2238           then
2239             begin
2240               switch := false;
2241               NewValTabLink".FirstDimen := NewDim
2242             end
2243             else DimPtr".NextDimen := NewDim;
2244               DimPtr := NewDim
2245             end;
2246           NewDim".NextDimen := nil;
2247           RightValPtr := RightArg".FirstValue; elements := 0;
2248           switch := true;
2249           while elements < ResultLength do
2250             begin ( duplicate right arg values into result values )
2251               elements := elements + 1; new(NewValues);
2252               if RightValPtr = nil
2253               ( extend right argument if necessary )
2254               then RightValPtr := RightArg".FirstValue;
2255               NewValues".RealVal := RightValPtr".RealVal;
2256               if switch
2257               then
2258                 begin
2259                   switch := false;
2260                   NewValTabLink".FirstValue := NewValues
2261                 end
2262                 else NewPtr".NextValue := NewValues;
2263                   NewPtr := NewValues;
2264                   RightValPtr := RightValPtr".NextValue
2265                 end;
2266               NewValues".NextValue := nil;
2267             end
2268           end ( reshape );
2269
2270
2271 procedure InnerProduct(LeftArg, RightArg: TypeValTabPtr);
2272
2273 var
2274   Inpro1Code, Inpro2Code, LeftSkip, RightSkip: integer;
2275   icount, jcount, kcount, lcount, mcount: integer;
2276   LastLeftDim, FirstRightDim, CommonLength: integer;
2277   lptr: "values;
2278   hold: real;
2279   SFloat, value: real;
2280
2281 begin ( inner product is matrix multiplication )
2282   DimPtr := LeftArg".FirstDimen;
2283   if LeftArg".FirstDimen <> nil then
2284     while DimPtr".NextDimen <> nil do
2285       DimPtr := DimPtr".NextDimen
2286       ( get last dimen of left arg(if any) );
2287   if (DimPtr <> nil) and (RightArg".FirstDimen <> nil)
2288   then
2289     if DimPtr".dimenlength <> RightArg".FirstDimen".dimenlength
2290     then
2291       error(52)
2292       ( last dim of left arg not = to first dim of right arg )
2293     else
2294       begin
2295         Inpro1Code := code div 100 ( separate operators );
2296         Inpro2Code := code - 100 * Inpro1Code;
2297         new(NewValTabLink);
2298         OldValTabLink".NextValTabLink := NewValTabLink;
2299         NewValTabLink".NextValTabLink := nil;
2300         NewValTabLink".IntermedResult := true;
2301         if not LeftArg".ForwardOrder
2302         then ReverseLinkList(LeftArg);
2303         if not RightArg".ForwardOrder
2304         then ReverseLinkList(RightArg);
2305         NewValTabLink".ForwardOrder := true;
2306         NewValTabLink".dimensions := LeftArg".dimensions +
2307         RightArg".dimensions - 2;
2308         if NewValTabLink".dimensions < 0
2309         then NewValTabLink".dimensions := 0;
2310         switch := true; LastLeftDim := 0;
2311         if LeftArg".FirstDimen <> nil
2312         then
2313           begin ( copy all but last of left arg dims into result )
2314             LeftSkip := 1; DimPtr := LeftArg".FirstDimen;
2315             while DimPtr".NextDimen <> nil do
2316               begin ( copy left arg dimensions )
2317                 new(NewDim);
2318                 NewDim".dimenlength := DimPtr".dimenlength;
2319                 LeftSkip := LeftSkip + DimPtr".dimenlength;
2320                 if switch
2321                 then
2322                   begin
2323                     switch := false;
2324                     NewValTabLink".FirstDimen := NewDim
2325                   end
2326                   else NewPtr".NextDimen := NewDim;
2327                     NewPtr := NewDim; DimPtr := DimPtr".NextDimen
2328                   end;
2329                 LastLeftDim := DimPtr".dimenlength
2330               end;
2331             if RightArg".FirstDimen <> nil
2332             then
2333               begin
2334                 ( copy all but first of right arg dims into result )
2335                 RightSkip := 1;
2336                 DimPtr := RightArg".FirstDimen".NextDimen;
2337                 while DimPtr <> nil do
2338                   begin ( copy right arg dimensions )
2339                     new(NewDim);
2340                     NewDim".dimenlength := DimPtr".dimenlength;
2341                     RightSkip := RightSkip + DimPtr".dimenlength;
2342                     if switch
2343                     then
2344                       begin
2345                         switch := false;
2346                         NewValTabLink".FirstDimen := NewDim
2347                       end
2348                       else NewPtr".NextDimen := NewDim;
2349                         NewPtr := NewDim; DimPtr := DimPtr".NextDimen
2350                       end
2351                     end;
2352                   if switch then NewValTabLink".FirstDimen := nil
2353                   else NewDim".NextDimen := nil;
2354                   if LeftArg".FirstValue = nil then LeftSkip := 0;
2355                   if RightArg".FirstValue = nil then RightSkip := 0;
2356                   switch := true;
2357                   if RightArg".FirstDimen <> nil
2358                   then FirstRightDim := RightArg".FirstDimen".dimenlength
2359                   else FirstRightDim := 0;
2360                   if FirstRightDim > LastLeftDim
2361                   then CommonLength := FirstRightDim
2362                   else CommonLength := LastLeftDim;
2363                   icount := 0; LeftValPtr := LeftArg".FirstValue;
2364                   while icount < LeftSkip do
2365                     begin ( loop for each row in left arg )
2366                       lptr := LeftValPtr ( hold start of row position );
2367                       jcount := 0;
2368                       while jcount < RightSkip do
2369                         begin ( loop for each column in right arg )
2370                           LeftValPtr := lptr;
2371                           RightValPtr := RightArg".FirstValue;
2372                           lcount := 0;
2373                           while lcount < jcount do
2374                             begin ( skip to starting value in right arg )
2375                               RightValPtr := RightValPtr".NextValue;
2376                               if RightValPtr = nil then
2377                                 RightValPtr := RightArg".FirstValue
2378                                 ( extend arg );
2379                               lcount := lcount + 1
2380                             end;
2381                             kcount := 0;
2382                             while kcount < CommonLength do
2383                               begin ( loop for each element in row/column )
2384                                 SFloat := RightValPtr".RealVal;
2385                                 DyadComp(SFloat, LeftValPtr".RealVal,
2386                                   Inpro2Code);
2387                                 value := SFloat;
2388                                 if kcount = 0
2389                                 then
2390                                   ( set identity value for first time through )
2391                                   case Inpro1Code of
2392                                     52, 53, 78: SFloat := 0.0;
2393                                     54, 55, 56, 77: SFloat := 1.0;
2394                                     71, 72, 73, 74, 75, 76: ( null case )
2395                                   end ( case )
2396                                 else SFloat := hold;
2397                                 DyadComp(SFloat, value, Inpro1Code);
2398                                 hold := SFloat ( save summer result );

```

```

2399 LeftValPtr := LeftValPtr.NextValue;
2400 if LeftValPtr = nil then
2401   LeftValPtr := LeftArg.FirstValue
2402   ( extend arg );
2403   mcount := 0;
2404   while mcount < RightSkip do
2405     begin ( skip to next value in right arg )
2406       mcount := mcount + 1;
2407       RightValPtr := RightValPtr.NextValue;
2408       if RightValPtr = nil
2409         then RightValPtr := RightArg.FirstValue;
2410       end;
2411       kcount := kcount + 1
2412     end;
2413   new(NewValues); NewValues.RealVal := SFloat;
2414   if switch
2415     then
2416       begin
2417         switch := false;
2418         NewValTabLink.FirstValue := NewValues
2419       end
2420     else NewValPtr.NextValue := NewValues;
2421         NewValPtr := NewValues; jcount := jcount + 1;
2422         end;
2423         icount := icount + 1
2424       end;
2425       if switch then NewValTabLink.FirstValue := nil
2426       else NewValues.NextValue := nil
2427     end
2428   end ( innerproduct );
2429
2430 procedure OuterProduce(LeftArg, RightArg: TypeValTabPtr);
2431 var
2432   OutProCode: integer;
2433   SFloat: real;
2434 begin
2435   OutProCode := code div 10; new(NewValTabLink);
2436   OldValTabLink.NextValTabLink := NewValTabLink;
2437   NewValTabLink.NextValTabLink := nil;
2438   NewValTabLink.IntermedResult := true;
2439   if not LeftArg.ForwardOrder
2440     then ReverseLinkList(LeftArg);
2441   if not RightArg.ForwardOrder
2442     then ReverseLinkList(RightArg);
2443   NewValTabLink.ForwardOrder := true;
2444   NewValTabLink.dimensions := LeftArg.dimensions + RightArg.
2445   dimensions;
2446   switch := true; DimPtr := LeftArg.FirstDimen;
2447   while DimPtr <> nil do
2448     begin ( copy left arg dimensions to result )
2449       new(NewDim); NewDim.dimenlength := DimPtr.dimenlength;
2450       if switch
2451         then
2452           begin
2453             switch := false; NewValTabLink.FirstDimen := NewDim
2454           end
2455         else NewPtr.NextDimen := NewDim;
2456             NewPtr := NewDim; DimPtr := DimPtr.NextDimen
2457           end;
2458       DimPtr := RightArg.FirstDimen;
2459       while DimPtr <> nil do
2460         begin ( copy dimensions of right arg to result )
2461           new(NewDim); NewDim.dimenlength := DimPtr.dimenlength;
2462           if switch
2463             then
2464               begin
2465                 switch := false; NewValTabLink.FirstDimen := NewDim
2466               end
2467             else NewPtr.NextDimen := NewDim;
2468                 NewPtr := NewDim; DimPtr := DimPtr.NextDimen
2469               end;
2470           if switch then NewValTabLink.FirstDimen := nil
2471           else NewDim.NextDimen := nil;
2472           switch := true; LeftValPtr := LeftArg.FirstValue;
2473           while LeftValPtr <> nil do
2474             begin
2475               RightValPtr := RightArg.FirstValue;
2476               while RightValPtr <> nil do
2477                 begin
2478                   SFloat := RightValPtr.RealVal;
2479                   DyadComp(SFloat, LeftValPtr.RealVal, OutProCode);
2480                   new(NewValues);
2481                   if switch
2482                     then
2483                       begin
2484                         switch := false;
2485                         NewValTabLink.FirstValue := NewValues
2486                       end
2487                     else NewValPtr.NextValue := NewValues;
2488                         NewValues.RealVal := SFloat; NewValPtr := NewValues;
2489                         RightValPtr := RightValPtr.NextValue
2490                       end;
2491                   LeftValPtr := LeftValPtr.NextValue
2492                 end;
2493               if switch then NewValTabLink.FirstValue := nil
2494               else NewValues.NextValue := nil
2495             end ( outerproduct );
2496           end
2497         end
2498       end

```

```

2500 procedure dyadic(LeftArg, RightArg: TypeValTabPtr);
2501 { operators with codes of 52 and higher }
2502 var
2503   compatible: Boolean;
2504   arg: TypeValTabPtr;
2505   SFloat: real;
2506 begin
2507   if code > 1000 then InnerProduct(LeftArg, RightArg)
2508   else
2509     if code > 100 then OuterProduce(LeftArg, RightArg)
2510     else
2511       if code > 80
2512       then
2513         case code of
2514           87: IndexOf(LeftArg, RightArg);
2515           88: reshape(LeftArg, RightArg);
2516           89: catenate(LeftArg, RightArg)
2517         end ( case )
2518       else
2519         begin ( simple dyadics )
2520           compatible := true;
2521           if (LeftArg.dimensions >= 1) and (RightArg.
2522             dimensions >= 1)
2523           then
2524             if LeftArg.dimensions <> RightArg.dimensions
2525             then
2526               compatible := false
2527               ( different ranks/neither scalar )
2528             else
2529               begin ( ranks match - check lengths )
2530                 LeftDimPtr := LeftArg.FirstDimen;
2531                 RighthDimPtr := RightArg.FirstDimen;
2532                 while LeftDimPtr <> nil do
2533                   begin
2534                     if LeftDimPtr.dimenlength <> RighthDimPtr.
2535                     dimenlength
2536                     then
2537                       compatible := false ( different length(s) );
2538                       LeftDimPtr := LeftDimPtr.NextDimen;
2539                       RighthDimPtr := RighthDimPtr.NextDimen
2540                     end
2541                   end;
2542                 if compatible
2543                 ( arguments suitable for dyadic operation )
2544               then
2545                 begin ( build dimensions of result )
2546                   if RightArg.dimensions > LeftArg.dimensions
2547                   then arg := RightArg
2548                   else
2549                     arg := LeftArg ( result has shape of larger arg );
2550                   new(NewValTabLink);
2551                   OldValTabLink.NextValTabLink := NewValTabLink;
2552                   NewValTabLink.NextValTabLink := nil;
2553                   NewValTabLink.IntermedResult := true;
2554                   if LeftArg.ForwardOrder <> RightArg.ForwardOrder
2555                   then ReverseLinkList(LeftArg);
2556                   NewValTabLink.ForwardOrder := arg.ForwardOrder;
2557                   NewValTabLink.dimensions := arg.dimensions;
2558                   switch := true; DimPtr := arg.FirstDimen;
2559                   while DimPtr <> nil do
2560                     begin ( copy dimensions to result )
2561                       new(NewDim);
2562                       NewDim.dimenlength := DimPtr.dimenlength;
2563                       if switch
2564                         then
2565                           begin
2566                             switch := false;
2567                             NewValTabLink.FirstDimen := NewDim
2568                           end
2569                         else NewPtr.NextDimen := NewDim;
2570                             NewPtr := NewDim;
2571                             DimPtr := DimPtr.NextDimen
2572                           end;
2573                       if switch
2574                         then
2575                           NewValTabLink.FirstDimen := nil
2576                           ( result is a scal )
2577                         else NewDim.NextDimen := nil;
2578                             switch := true;
2579                             RightValPtr := RightArg.FirstValue;
2580                             LeftValPtr := LeftArg.FirstValue;
2581                             ValPtr := arg.FirstValue;
2582                             while ValPtr <> nil do
2583                               begin ( perform operation )
2584                                 new(NewValues);
2585                                 SFloat := RightValPtr.RealVal;
2586                                 DyadComp(SFloat, LeftValPtr.RealVal, code);
2587                                 NewValues.RealVal := SFloat;
2588                                 if switch
2589                                   then
2590                                     begin
2591                                       switch := false;
2592                                       NewValTabLink.FirstValue := NewValues
2593                                     end
2594                                   else NewValPtr.NextValue := NewValues;
2595                                       NewValPtr := NewValues;
2596                                       ValPtr := ValPtr.NextValue;
2597                                       LeftValPtr := LeftValPtr.NextValue;
2598                                     end
2599                                 end

```

```

2601 RightValPtr := RightValPtr.NextValue;
2602 if LeftValPtr = nil then
2603   LeftValPtr := LeftArg.FirstValue
2604   ( extend arg );
2605   if RightValPtr = nil then
2606     RightValPtr := RightArg.FirstValue
2607     ( extend );
2608   end;
2609   if switch
2610   then
2611     NewValTabLink.FirstValue := nil
2612     ( vector of len 0 )
2613   else NewValues.NextValue := nil
2614   end
2615   else
2616     error(55)
2617     ( arguments incompatible for dyadic operation )
2618   end
2619 end ( dyadic );
2620
2621 procedure FunCall(var ValidFunk: Boolean);
2622 var
2623   ValidPm: Boolean;
2624 begin ( funcall )
2625   ValidFunk := false;
2626   if FunctCall
2627   then
2628     begin
2629       if TokenTabPtr.noun <> StatEnd
2630       then
2631         begin
2632           SubrTabPtr.TokenCallingSubr := TokenTabPtr;
2633           primary(ValidPm); if not ValidPm then error(17);
2634           ( 'leftarg of dyadic func call not a primary' )
2635         end;
2636         CallSubr; ValidFunk := true;
2637       end;
2638     end ( funcall );
2639
2640 begin ( expression )
2641   primary(ValidPri);
2642   if not ValidPri
2643   then
2644     begin
2645       if TokenTabPtr.noun = StatEnd
2646       then begin ValidExp := true; assign1 := true end
2647       else ValidExp := false
2648       end
2649     else
2650     begin
2651       DoneExp := false;
2652       while not DoneExp do
2653       begin
2654         FunCall(ValidFunc);
2655         if ValidFunc
2656         then begin expression(ValidExp); DoneExp := true end
2657         else
2658         begin
2659           assignment(ValidAssn);
2660           if ValidAssn and (TokenTabPtr.noun = StatEnd)
2661           then begin DoneExp := true; ValidExp := true; end;
2662           if not ValidAssn
2663           then
2664             if mop
2665             then
2666               begin
2667                 monadic(OperTabPtr.OperPtr, hold);
2668                 OperTabPtr.OperPtr := NewValTabLink
2669               end
2670             else
2671             if not dop
2672             then begin ValidExp := true; DoneExp := true end
2673             else
2674             begin
2675               primary(ValidPri);
2676               if not ValidPri
2677               then
2678                 error(13)
2679                 ( dyad oper not preceded by a pri )
2680             else
2681             begin
2682               dyadic(OperTabPtr.OperPtr, OperTabPtr.
2683                 LastOper.OperPtr);
2684               AuxOperTabPtr := OperTabPtr;
2685               OperTabPtr := OperTabPtr.LastOper;
2686               PtrLastOper := OperTabPtr;
2687               dispose(AuxOperTabPtr);
2688               OperTabPtr.OperPtr := NewValTabLink;
2689             end;
2690           end;
2691         end;
2692       end;
2693     end;
2694   end;
2695 end ( expression );
2696
2697 end ( expression );
2698
2699
2700

```

```

2701 begin ( parser )
2702 assign := false; assign1 := false; DoneParse := false;
2703 repeat
2704   expression(ValidExp) ( checks for valid expression );
2705   if not ValidExp then error(10) ( 'invalid expression' )
2706   else
2707     if SpecSymbol(XRightArrow)
2708     then
2709       if not ((OperTabPtr.OperPtr.FirstValue = nil) and (
2710         OperTabPtr.OperPtr.dimensions > 0))
2711       then ( branch )
2712       ( result of expression is at opertabptr )
2713       if OperTabPtr.OperPtr.FirstValue.RealVal - 1.0 * trunc
2714         (OperTabPtr.OperPtr.FirstValue.RealVal) <> 0.0
2715       then error(12) ( stmt.num.to branch to not an integer )
2716       else
2717         if SubrTabPtr = nil
2718         then
2719           begin ( function mode )
2720             TokenTabPtr := hold; DoneParse := true
2721           end
2722         else
2723           if trunc(OperTabPtr.OperPtr.FirstValue.RealVal) in
2724             [1 .. (SubrTabPtr.CalledSubr.NumOfStatements)]
2725           then
2726             begin
2727               VFuncHold := SubrTabPtr.CalledSubr.FirstStatement;
2728               for cnt := 1 to trunc(OperTabPtr.OperPtr.
2729                 FirstValue.RealVal) do
2730                 begin
2731                   VFuncPtr := VFuncHold;
2732                   TokenTabPtr := VFuncPtr.NextStmnt;
2733                   VFuncHold := VFuncPtr.NextVFuncPtr
2734                 end;
2735                 AuxOperTabPtr := OperTabPtr;
2736                 OperTabPtr := OperTabPtr.LastOper;
2737                 dispose(AuxOperTabPtr); PtrLastOper := OperTabPtr;
2738                 TokenTabPtr := VFuncPtr.NextStmnt
2739               end
2740             else ReturnToCallingSubr
2741             else ( successor )
2742             else ( successor )
2743             begin
2744               if not assign1 then OutPutVal; assign1 := false;
2745               if SubrTabPtr = nil
2746               then
2747                 begin ( interpretive )
2748                   hold := TokenTabPtr;
2749                   TokenTabPtr := TokenTabPtr.NextToken;
2750                   DoneParse := true
2751                 end
2752             else ( function )
2753             begin
2754               VFuncPtr := VFuncPtr.NextVFuncPtr;
2755               DoneSuccessor := false;
2756               repeat
2757                 if VFuncPtr <> nil
2758                 then
2759                   begin
2760                     TokenTabPtr := VFuncPtr.NextStmnt;
2761                     DoneSuccessor := true
2762                   end
2763                 else
2764                 begin
2765                   ReturnToCallingSubr;
2766                   if TokenTabPtr.noun = StatEnd
2767                   then DoneSuccessor := true;
2768                 end;
2769               until DoneSuccessor;
2770             end;
2771           end
2772         until DoneParse;
2773         release ( release memory );
2774       end ( parser );
2775
2776 begin ( scanner )
2777 initializeCharacterSet; ReadInErrorMsgs;
2778 InitParser ( initialize tables etc. ); FillUpTables;
2779 FunctionMode := false; FirstFunction := true;
2780 OldValTabLink := nil; OldFuncTabPtr := nil; OldVarTabPtr := nil;
2781 OldTokenPtr := nil; NewTokenPtr := nil; NewFuncTabPtr := nil;
2782 NewVFuncPtr := nil; HoldTokenPtr := nil; TokenError := false;
2783 NewValTabLink := nil; NewVarTabPtr := nil; GetAPLstatement;
2784 while (APLstatement[1] <> character[ForwardSlash]) or (APLstatement[2]
2785   <> character[Lasterisk]) do ( ' ends program ' )
2786 begin
2787   SkipSpaces; TokenSwitch := true;
2788   while (position <= LineLength) and (not TokenError) and (not
2789     LineTooLong) do
2790     begin ( scanning )
2791       if APLstatement[position] = character[del]
2792       ( function delimiter )
2793       then ( del encountered )
2794       ( if functionMode
2795         then
2796           begin ( end of current function )
2797             if NewFuncTabPtr <> nil
2798             then NewFuncTabPtr.NumOfStatements := FuncStatements;
2799             if FuncStatements > 0

```


"Don't Fail Me Now"

By Srully Blotnick

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 play a 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 way 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."

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.

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. Bruell (John Wiley, \$26.95); and *Data Structures Using Pascal* by A. Tenenbaum 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 Queueing 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.

PUG

Computer Generated Population Pyramids Using Pascal

Gerald R. Pitzl
Geography Department
Macalester College
St. Paul, Minnesota

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.¹ Virtually all of the programs, however, are written in FORTRAN and are suitable for easy installation primarily on large mainframe computers.²

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.³

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.⁴

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.⁵ 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 later category; the less developed in the former.

The Programs

Three programs have been developed for student use in an introductory human geography course.⁶ 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 sequence described 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 pyramid_file(input, output);
(a program to create an external file of
population pyramid data)

const
separator = '-----';

type
data =
record
country: packed array [1..15] of char;
year: packed array [1..4] of char;
malepercent, femalepercent: array [1..16] of integer;
end;
identifiers = file of data;

var
temp: data;
info: identifiers;
filename: packed array [1..10] of char;
answer: char;

procedure read_data;
var
i: integer;
begin
writeln;
write(' enter file name: ');
readln(filename);
writeln;
write(' enter place name -- use 15 columns: ');
readln(temp.country);
writeln;
write(' enter the year of the data: ');
readln(temp.year);
writeln;
write(' now, enter male and female percentages for');
write(' each age group; enter male percentages first');
write(' from the highest age group to the lowest. ');
write(' you must have 16 entries for each group. ');
write(' enter as integers all on the same line. ');
writeln;
for i := 1 to 16 do
read(temp.malepercent[i]);
write(' next, enter the female percentages. ');
for i := 1 to 16 do
read(temp.femalepercent[i]);
readln;
end (read_data);

procedure echo_data;
var
i: integer;
begin
writeln;
write(separator);
writeln;
write(' the following information was entered: ');
writeln;
write(' ', temp.country);
writeln;
write(' ', temp.year);
writeln;
write(' male : ');
for i := 1 to 16 do
write(temp.malepercent[i]);
writeln;
write(' female : ');
for i := 1 to 16 do
write(temp.femalepercent[i]);
writeln;
write(separator);
write(' is the information correct? ');
write(' if yes, enter a "y"; if not, enter ');
write(' "n" ');
readln(answer);
writeln;
end (echo_data);

procedure store_data;
var
i: integer;
begin
re-write(info, filename);
info.country := temp.country;
put(info);
info.year := temp.year;
put(info);
for i := 1 to 16 do
begin
info.malepercent[i] := temp.malepercent[i];
put(info);
end;
for i := 1 to 16 do
begin
info.femalepercent[i] := temp.femalepercent[i];
put(info);
end;
close(info);
end (store_data);

```

```

begin
read_data;
echo_data;
if (answer = 'y') or (answer = 'Y') then
begin
store_data;
writeln(' **** operation completed **** ');
writeln(' information stored in the file: ', filename);
end
else
writeln(' 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 data file
created by the program, "pyramid_file", and
to make changes if necessary)

type
data =
record
country: packed array [1..15] of char;
year: packed array [1..4] of char;
mpct, fpct: array [1..16] of integer;
end;
identifiers = file of data;

var
temp: data;
info: identifiers;
filename: packed array [1..10] of char;

procedure skip_lines;
var
i: integer;
begin
for i := 1 to 10 do
writeln;
end (skip_lines);

procedure access_file_data;
var
i: integer;
begin
write(' enter file name: ');
readln(filename);
reset(info, filename);
temp.country := info.country;
get(info);
temp.year := info.year;
get(info);
for i := 1 to 16 do
begin
temp.mpct[i] := info.mpct[i];
get(info);
end;
for i := 1 to 16 do
begin
temp.fpct[i] := info.fpct[i];
get(info);
end;
close(info);
end (access_file_data);

procedure publish_data;
const
separator = '-----';
var
i: integer;
begin
skip_lines;
write(separator);
writeln;
write(' the following information is ');
write(' contained in ', filename, ' ');
writeln;
write(' ', temp.country);
writeln;
write(' ', temp.year);
writeln;
write(' male percent: ');
for i := 1 to 16 do
write(temp.mpct[i]);
writeln;
write(' female percent: ');
for i := 1 to 16 do
write(temp.fpct[i]);
writeln;
write(separator);

```

```

write(separator)
end (publish_data) ;

procedure make_file_changes;

var
  i: integer;
  selector: char;

begin
  write;
  write;
  write(" you may make any number of changes");
  write(" by selecting the appropriate symbol");
  write(" for the data to be changed.");
  write;
  write(" use the following set of selectors");
  write;
  write(" area name == 'A'");
  write(" year == 'Y'");
  write(" male percent == 'M'");
  write(" female percent == 'F'");
  write;
  write(" enter the selector, then <cr>, and");
  write(" you will be prompted to enter the");
  write(" new data.");
  write;
  write(" when you have completed the changes.");
  write(" enter an 'e' to end the session.");
  write;
  write(" enter a selector:  ");
  read(selector);
  repeat
    case selector of
      'A', 'A':
        begin
          write(" enter new area name.");
          write(" use 15 columns:  ");
          read(temp.country);
          end;
        'Y', 'Y':
          begin
            write(" enter the new years:  ");
            read(temp.year);
            end;
        'M', 'M':
          begin
            write(" enter all sixteen male percent values --");
            write(" highest one groups to lowest:");
            for i := 1 to 16 do
              read(temp.mpcct[i]);
            end;
            read;
            end;
        'F', 'F':
          begin
            write(" enter all sixteen female percent values --");
            write(" highest one groups to lowest:");
            for i := 1 to 16 do
              read(temp.fpcct[i]);
            end;
            read;
            end;
    end (case) ;
    write(" make another selection:  ");
    read(selector);
  until (selector = 'e') or (selector = 'e');
  rewrite(info, filename);
  info.country := temp.country;
  put(info);
  info.year := temp.year;
  put(info);
  for i := 1 to 16 do
    begin
      info.mpcct[i] := temp.mpcct[i];
      put(info);
    end;
  for i := 1 to 16 do
    begin
      info.fpcct[i] := temp.fpcct[i];
      put(info);
    end;
  close(info);
  write(" new data stored in ", filename);
  write;
  publish_data;
end (make_file_changes) ;

procedure modify_file_choices;

var
  choices: char;

begin
  write;
  write;
  write(" do you want to modify the data? --");
  write(" if so, enter a 'Y'");
  write(" if not, enter an 'N'");
  read(choices);
  if (choice = 'Y') or (choice = 'y') then
    make_file_changes;
  else
    write(" no changes to the file.");
  end (modify_file_choices) ;

begin
  access_file_data;
  publish_data;
  modify_file_choice;
end.

```

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 there 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.pyr

-----
the following information is
contained in sweden.pyr:

sweden
1970

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

-----

do you want to modify the data? --
if so, enter a 'y'
if not, enter an 'n':  n

no changes to the file.

Ready

program drawpyramid(input, output);
(* program to produce a population pyramid graphic)

const
  blank = ' ';

type
  data =
    record
      country: packed array [1..15] of char;
      year: packed array [1..4] of char;
      male, female: array [1..16] 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;
  pyramids: 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;
    end (initialize_arrays) ;

procedure plot_choices; (* single plot or superimposed plot)

begin
  write;
  write(" enter a 'd' if this is a double plot;");
  write(" enter an 's' for a single plot:  ");
  read(choice);
  write;
  end (plot_choices) ;

procedure enter_file_names;

begin
  if (choice = 'd') or (choice = 'd') then (*double plot)
  begin
    write(" enter each file name on a separate line.");
    write(" use ten columns for each; note the marker, '****';");
    write(" if you are at the line printer,");
    write(" position the writing head to the");
    write(" last line of the paper before");
    write(" entering <cr> after the second file name.");
    write;
    write("****");
    read(file1);
    read(file2);
  end;

```

```

end
else (single plot)
begin
writein(' enter the file name using ten columns:');
writein(' note the "arker"');
writein(' if you are at the line printer,');
writein(' position the writing head to the');
writein(' last line of the paper before');
writein(' entering <cr>');
writein();
writein(' ');
readln(file1)
end
end (enter_title_name) ;

```

procedure labels;

```

var
i, j, k: integer;
shorttitle: packed array [1..18] of char;
longtitle: packed array [1..35] of char;
agegroups: packed array [1..65] of char;
newborn: packed array [1..10] of char;

begin
for i := 2 to 42 do
begin
matrix[i, 1] := '1';
matrix[i, 63] := '1';
end;
for i := 1 to 63 do
begin
matrix[i, 1] := '-';
matrix[42, i] := '-';
end;
for i := 6 to 37 do
matrix[i, 27] := '1';
for i := 7 to 47 do
matrix[37, i] := '-';
if (choice = 'n') or (choice = 'd') then
begin
longtitle := 'population pyramids -- superimposed';
for i := 3 to 37 do
matrix[i, 5] := longtitle[i - 2]
end
else
begin
shorttitle := 'population pyramid';
for j := 3 to 20 do
matrix[j, 5] := shorttitle[j - 2]
end;
newborn := 'malefemale';
for j := 13 to 16 do
matrix[13, j] := newborn[j - 12];
for j := 38 to 43 do
matrix[13, j] := newborn[j - 33];
matrix[4, 53] := 'a';
matrix[4, 54] := 'o';
matrix[4, 55] := 'e';
matrix[6, 53] := '7';
matrix[6, 54] := '5';
matrix[6, 55] := '4';
agegroups :=
'70-7465-6960-6455-6950-5445-4940-4435-3930-3425
-2920-2415-1910-14
';
i := 1;
k := 8;
while i <= 65 do
begin
for j := 52 to 56 do
begin
matrix[k, j] := agegroups[i];
i := i + 1;
end;
k := k + 2;
end;
matrix[34, 53] := '5';
matrix[34, 54] := '-';
matrix[34, 55] := '9';
matrix[36, 53] := '0';
matrix[36, 54] := '-';
matrix[36, 55] := '4';
j := 7;
while i <= 47 do
begin
matrix[38, i] := ' ';
j := j + 4;
end;
matrix[38, 27] := '0';
matrix[38, 71] := '0';
matrix[38, 81] := '0';
matrix[39, 11] := '8';
matrix[39, 15] := '6';
matrix[39, 19] := '4';
matrix[39, 23] := '2';
matrix[39, 31] := '2';
matrix[39, 35] := '4';
matrix[39, 39] := '6';
matrix[39, 43] := '8';
matrix[39, 46] := '1';
matrix[39, 47] := '0';
matrix[41, 24] := '0';
matrix[41, 25] := '0';
matrix[41, 26] := '1';
matrix[41, 27] := '0';
matrix[41, 28] := '0';
matrix[41, 29] := '0';
matrix[41, 30] := '1';
end (labels) ;

```

procedure retrieve_and_ession_data;

```

var
i, j, k: integer;
filearray: array [1..2] of filename;

begin
if (choice = '0') or (choice = 'd') then (double plot)
begin
filearray[1] := file1;
filearray[2] := file2;
for i := 1 to 2 do
begin
if i = 1 then
k := 43;
else
k := 111;
reset(pyramid, filearray[i]);
temp.country := pyramid.country;
get(pyramid);
if i = 2 then
for i := 1 to 15 do
matrix[4, i + 17] := temp.country[i];
else
for i := 1 to 15 do
matrix[4, i + 2] := temp.country[i];
temp.year := pyramid.year;
get(pyramid);
if i = 2 then
for j := 1 to 4 do
matrix[5, i + 17] := temp.year[j];
else
for i := 1 to 4 do
matrix[5, i + 2] := temp.year[i];
for i := 1 to 16 do
begin
temp.male[i] := pyramid.male[i];
get(pyramid);
if matrix[2 * i + 4, 27 + 2 * temp.male[i]] <> blank then
matrix[2 * i + 4, 27 + 2 * temp.male[i]] := 'm';
else
matrix[2 * i + 4, 27 + 2 * temp.male[i]] := chr(k);
end;
for j := 1 to 16 do
begin
temp.female[j] := pyramid.female[j];
get(pyramid);
if matrix[2 * i + 4, 27 + 2 * temp.female[j]] <> blank then
matrix[2 * i + 4, 27 + 2 * temp.female[j]] := 'm';
else
matrix[2 * i + 4, 27 + 2 * temp.female[j]] := chr(k);
end;
close(pyramid);
end
end
else (single plot)
begin
reset(pyramid, file1);
temp.country := pyramid.country;
get(pyramid);
for i := 1 to 15 do
matrix[4, i + 2] := temp.country[i];
temp.year := pyramid.year;
get(pyramid);
for i := 1 to 4 do
matrix[5, i + 2] := temp.year[i];
for i := 1 to 16 do
begin
temp.male[i] := pyramid.male[i];
get(pyramid);
matrix[2 * i + 4, 27 + 2 * temp.male[i]] := 'm';
end;
for i := 1 to 16 do
begin
temp.female[i] := pyramid.female[i];
get(pyramid);
matrix[2 * i + 4, 27 + 2 * temp.female[i]] := 'm';
end;
close(pyramid);
end
end (retrieve_and_ession_data) ;

```

procedure symbol_explanation;

```

var
check1, check2: data;

begin
writein();
reset(pyramid, file1);
check1.country := pyramid.country;
get(pyramid);
check1.year := pyramid.year;
close(pyramid);
reset(pyramid, file2);
check2.country := pyramid.country;
get(pyramid);
check2.year := pyramid.year;
close(pyramid);
if check1.country = check2.country then (same area for both plots)
begin
writein(' ', check1.year; 4, ' = 4');
writein(' ', check2.year; 4, ' = 0');
writein(' "a": same value for each year')
end
else (different areas)
begin
writein(' ', check1.country, ' = 4');
writein(' ', check2.country, ' = 0');
writein(' "a": same value for both areas')
end
end (symbol_explanation) ;

```

procedure skiplines; (for proper output formatting)

```

var
i: integer;

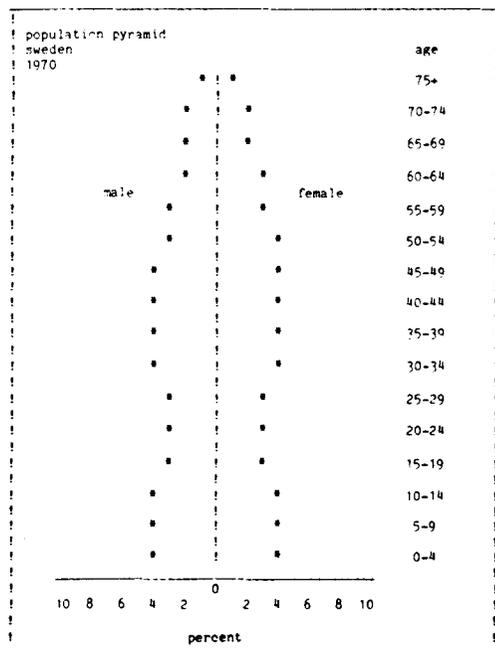
```

```

begin
  for i := 1 to 12 do
    writeln
  end (skirlines) ;

procedure produce_pyramid;
var
  i, j: integer;
begin
  skirlines;
  for i := 1 to 42 do
    for j := 1 to 63 do
      begin
        write(matrix[i, j]);
        if j = 63 then
          writeln
        end;
        if (choice = 't') or (choice = 'd') then
          symbol_explanation;
        skiplines
      end (produce_pyramid) ;
    end;
  end;
begin
  initialize_arrays;
  plot_choices;
  enter_file_name;
  labels;
  retrieve_and_assign_data;
  produce_pyramid;
end.

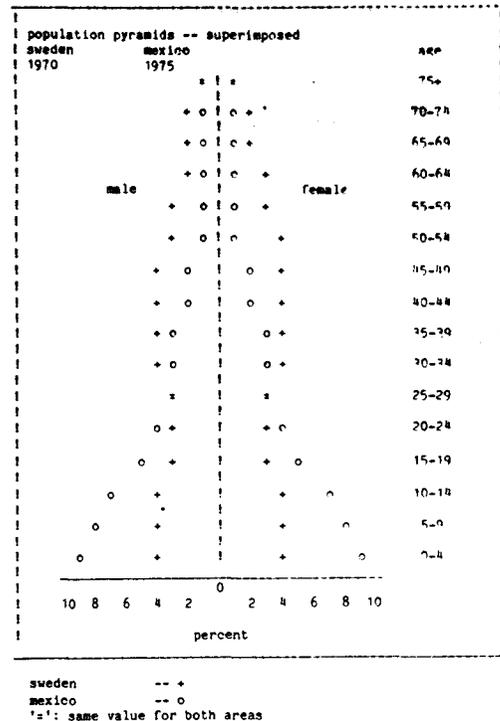
```



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 48824

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.

3. Mark S. Mormonier, *Computer-Assisted Cartography: Principles and Prospects* (Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1982), p. 22.

4. John L. Kirkley, Editor, "Our Industry Could Lead a Liberal Arts Renaissance," *Datamation*, March, 1983, p. 29.

5. Brian J. L. Berry, et al, *The Geography of Economic Systems* (Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1976), p.36.

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

PUG

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 NAS1-16985 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 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 (PROCE-

DURESSs, 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 opera-

tion). 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.

PROCESSEs in Path Pascal can either be normal or INTERRUPT PROCESSEs. An INTERRUPT PROCESS has two special attributes not associated with normal PROCESSEs. 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 those 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" an 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.

```
PROGRAM ISLAND: (* WOLF AND RABBIT SIMULATION PROGRAM *)
CONST
  BEL      7:  (* ASCII FOR TERMINAL BELL *)
  CR       13: (* ASCII FOR TERMINAL END OF INPUT *)
  ESC      27: (* ASCII FOR TERMINAL CREATOR *)

  CRTSZ    120: (* SIZE OF TERMINAL BUFFER *)

  XMAX     26: (* ISLAND SIZE: COLS+1 *)
  YMAX     18: (* ISLAND SIZE: ROWS+1 *)

  XWOLF    10: (* INITIAL COLUMN FOR WOLVES *)
  XRABBIT  16: (* INITIAL COLUMN FOR RABBITS *)

TYPE
  CRTPTR = CRTOBJ;
  CRTOBJ = OBJECT (* SYNCHRONIZE TERMINAL INPUT/OUTPUT *)

    PATH
      1:(PUSH); (* ONE AT A TIME *)
      1:(POP);  (* ONE AT A TIME *)
      CRTSZ:(PUSH:POP) (* PUSH THEN POP *)
    END;

  VAR
    BUF : ARRAY [1..CRTSZ] OF CHAR;
    INPTR : INTEGER;
    OUTPTR : INTEGER;

  ENTRY PROCEDURE PUSH(CH:CHAR);
  BEGIN
    BUF[INPTR] := CH;
    IF INPTR=CRTSZ THEN
      INPTR := 1;
    ELSE
      INPTR := INPTR+1;
    END; (* PROCEDURE PUSH *)

  ENTRY PROCEDURE POP(VAR CH:CHAR);
  BEGIN
    CH := BUF[OUTPTR];
    IF OUTPTR=CRTSZ THEN
      OUTPTR := 1;
    ELSE
      OUTPTR := OUTPTR+1;
    END; (* PROCEDURE POP *)

  INIT:
  BEGIN
    INPTR := 1;
    OUTPTR := 1;
  END;
END: (* OBJECT CRTOBJ *)

VAR
  CRTIBUF : CRTPTR; (* INPUT BUFFER *)
  CRTOBUF : CRTPTR; (* OUTPUT BUFFER *)
  WINIT : INTEGER; (* WOLF: INITIAL ENERGY *)
```

```

WYRS : INTEGER: (* WOLF: NORMAL LIFETIME *)
WANNIAL : INTEGER: (* WOLF: ANNUAL ENERGY USAGE *)
WPFERTILE : INTEGER: (* WOLF: ENERGY FOR FERTILITY *)
WCRHOWD : INTEGER: (* WOLF: MAX ENERGY OF NEIGHBORS *)
WMAX : INTEGER: (* WOLF: MAX ENERGY FOR AN INDIVIDUAL *)

RINIT : INTEGER: (* RABBIT: INITIAL ENERGY *)
RYRS : INTEGER: (* RABBIT: NORMAL LIFETIME *)
RANNIAL : INTEGER: (* RABBIT: ANNUAL ENERGY USAGE *)
RPFERTILE : INTEGER: (* RABBIT: ENERGY FOR FERTILITY *)
RCRHOWD : INTEGER: (* RABBIT: MAX ENERGY OF NEIGHBORS *)
RMAX : INTEGER: (* RABBIT: MAX ENERGY FOR AN INDIVIDUAL *)

CREATOR : OBJECT (* SYNCHRONIZE WOLF/RABBIT CREATION *)
  PATH
  1:(CREATE:STARTUP)
  END:
  VAR
  XX : INTEGER:
  YY : INTEGER:
  EE : INTEGER:
  ENTRY PROCEDURE CREATE(X,Y,E:INTEGER):
  BEGIN
  XX := X:
  YY := Y:
  EE := E:
  END: (* PROCEDURE CREATE *)
  ENTRY PROCEDURE STARTUP(VAR X,Y,E:INTEGER):
  BEGIN
  X := XX:
  Y := YY:
  E := EE:
  END: (* PROCEDURE STARTUP *)
END: (* OBJECT CREATOR *)

SCREEN : OBJECT (* COORDINATE SCREEN *)
  PATH
  1:(SETUP,KILL,LOOK,ASSIGN,CHANGE,DONE)
  END:
  VAR
  STOP : BOOLEAN:
  INFO : ARRAY [0..XMAX,0..YMAX] OF INTEGER:
  ENTRY PROCEDURE SETUP: (* (RE)INITIALIZATION *)
  VAR
  X : INTEGER:
  Y : INTEGER:
  BEGIN
  STOP := FALSE:
  FOR X := 0 TO XMAX DO (* RESET ENERGIES *)
  FOR Y := 0 TO YMAX DO
  INFO(X,Y) := 0:
  END: (* PROCEDURE SETUP *)
  ENTRY PROCEDURE KILL: (* BEGIN TERMINATION *)
  BEGIN
  STOP := TRUE:
  END: (* PROCEDURE KILL *)
  ENTRY PROCEDURE LOOK(X,Y:INTEGER; VAR ER,EW:INTEGER):
  PROCEDURE TEST(ENERGY:INTEGER):
  BEGIN
  IF ENERGY=0 THEN
  ER := ER+ENERGY
  ELSE
  EW := EW+ENERGY:
  END: (* PROCEDURE TEST *)
  BEGIN
  ER := 0: (* SURROUNDING RABBIT ENERGIES *)
  EW := 0: (* SURROUNDING WOLF ENERGIES *)
  TEST(INFO(X-1,Y)):
  TEST(INFO(X+1,Y)):
  TEST(INFO(X,Y-1)):
  TEST(INFO(X,Y+1)):
  END: (* PROCEDURE LOOK *)
  PROCEDURE WRITES(X,Y,E:INTEGER): (* WRITE TO SCREEN *)
  BEGIN
  CRTORBUF^.PUSH(CHR(ESC)): (* VT52 JUMP *)
  CRTORBUF^.PUSH('Y'):
  CRTORBUF^.PUSH(CHR(Y+31)):
  CRTORBUF^.PUSH(CHR(X+31)):
  IF E=0 THEN (* EMPTY *)
  CRTORBUF^.PUSH(' ')
  ELSE
  IF E<0 THEN (* RABBIT *)
  CRTORBUF^.PUSH('R')
  ELSE (* WOLF *)
  CRTORBUF^.PUSH('W'):
  END: (* PROCEDURE WRITES *)
  ENTRY PROCEDURE ASSIGN(X,Y,E:INTEGER):
  BEGIN
  INFO(X,Y) := E: (* UPDATE IN-MEMORY *)
  WRITES(X,Y,E): (* UPDATE SCREEN *)
  END: (* PROCEDURE ASSIGN *)
  ENTRY FUNCTION CHANGE(X,Y,E:INTEGER):BOOLEAN:
  BEGIN
  CHANGE := FALSE:
  IF (INFO(X,Y)=0) AND NOT STOP THEN (* FREE *)
  BEGIN
  INFO(X,Y) := E: (* UPDATE IN-MEMORY *)
  WRITES(X,Y,E): (* UPDATE SCREEN *)
  CHANGE := TRUE:
  END:
  END: (* FUNCTION CHANGE *)
  ENTRY FUNCTION DONE(X,Y,E:INTEGER):BOOLEAN:
  BEGIN
  INFO(X,Y) := E: (* UPDATE IN-MEMORY *)
  DONE := FALSE:
  IF (E=0) OR STOP THEN (* TERMINATE WOLF/RABBIT *)
  BEGIN
  WRITES(X,Y,0): (* UPDATE SCREEN *)
  DONE := TRUE:
  END:
  END: (* FUNCTION DONE *)
END: (* OBJECT SCREEN *)

SHUTDOWN : OBJECT (* SYNCHRONIZE TERMINATION *)
  PATH
  1:(SIGNAL:WAIT)
  END:
  ENTRY PROCEDURE SIGNAL:
  BEGIN
  END: (* PROCEDURE SIGNAL *)
  ENTRY PROCEDURE WAIT:
  BEGIN
  END: (* PROCEDURE WAIT *)
END: (* OBJECT SHUTDOWN *)

INTERRUPT PROCESS DLVJIN [PRIORITY=1,VECTOR=8300] (IRUP:CRTPTR):
(* DEFAULT VECTOR = 8300 *)
(* DEFAULT ADDRESS = 87777560 *)
VAR
CSR(87777560) : INTEGER:
RUF(87777562) : INTEGER:
CHRUP : INTEGER:
CH : CHAR:
BEGIN
REPEAT
SUBSET:
CSR := 64:
DOIO:
CHRUP := RUF:
CSR := CSR-64:
SHDRTN:
IF CHRUP=0 THEN
(IRUP^.PUSH(CHR(ESC))):
(IRUP^.PUSH(CHR(CHRUP MOD 128))):
UNTIL FALSE:
END: (* INTERRUPT PROCESS DLVJIN *)

INTERRUPT PROCESS DLVJOUT[PRIORITY=1,VECTOR=8320] (ORUP:CRTPTR):
(* DEFAULT VECTOR = 8320 *)
(* DEFAULT ADDRESS = 87777564 *)
VAR
ORI(87777564) : INTEGER:
RUI(87777566) : INTEGER:
OR : CHAR:
BEGIN
REPEAT
ORUP^.POP(OR):
SUBSET:
CSR := 64:
RUF := ORI(OR):
DOIO:
CSR := CSR-64:
SHDRTN:
UNTIL FALSE:
END: (* INTERRUPT PROCESS DLVJOUT *)

PROCESS WOLF(X,Y,ENERGY:INTEGER): (* ONE INSTANCE PER WOLF *)
VAR
ENERGY : INTEGER: (* CURRENT ENERGY LEVEL *)
ER : INTEGER: (* ENERGY LEVEL OF NEIGHBORING RABBITS *)
EW : INTEGER: (* ENERGY LEVEL OF NEIGHBORING WOLVES *)
DLY : INTEGER: (* "REST" TIME BETWEEN ACTIVITIES *)
BEGIN
ENERGY := ENERGY: (* ALWAYS 0 *)
REPEAT
ENERGY := ENERGY-WANNIAL: (* ANNUAL ENERGY USAGE *)
IF ER<0 THEN (* CHECK NEIGHBORS *)
ENERGY := ENERGY+ER: (* RABBITS TO EAT *)
ELSE
IF EW=WRHOWD THEN (* TOO MANY WOLVES *)
ENERGY := ENERGY-WANNIAL:
ELSE
IF (EW=0) AND (ENERGY-WPFERTILE) THEN (* REPRODUCE *)
REPRODUCE(CREATE(X,Y,ENERGY)):
IF ENERGY=0 THEN (* AVOID BECOMING A RABBIT *)
ENERGY := 0:
IF ENERGY-WMAX THEN (* CANNOT USE EXCESS ENERGY *)
ENERGY := WMAX:
DLY := WMAX-ENERGY: (* REST BASED UPON ENERGY *)
IF DLY=ENERGY THEN
DLY := ENERGY:
DELAY(DLY):
UNTIL SCREEN DONE(X,Y,ENERGY): (* DECIDE WHETHER STILL "ALIVE" *)
END: (* PROCESS WOLF *)

PROCESS RABBIT(X,Y,ENERGY:INTEGER): (* ONE INSTANCE PER RABBIT *)
VAR
ENERGY : INTEGER: (* CURRENT ENERGY LEVEL *)
ER : INTEGER: (* ENERGY LEVEL OF NEIGHBORING RABBITS *)
EW : INTEGER: (* ENERGY LEVEL OF NEIGHBORING WOLVES *)
DLY : INTEGER: (* "REST" TIME BETWEEN ACTIVITIES *)
BEGIN
ENERGY := ENERGY: (* ALWAYS 0 *)
REPEAT
ENERGY := ENERGY-WANNIAL: (* ANNUAL ENERGY USAGE *)
IF ER<0 THEN (* CHECK NEIGHBORS *)
ENERGY := ENERGY+ER: (* EATEN BY WOLF *)
ENERGY := 0:
ELSE
IF (EW=0) AND (ENERGY-WPFERTILE) THEN (* REPRODUCE *)

```

```

BEGIN
  CREATEP(CREATE(X,Y,ENERGY));
  IF EP = R(ROW) THEN (** RABBITS LOVE A CROWD **)
    ENERGY := ENERGY*2*ANNUAL;
  END;
  IF ENERGY < 0 THEN (** AVOID BECOMING A WOLF **)
    ENERGY := 0;
  IF ENERGY > RMAX THEN (** CANNOT USE EXCESS ENERGY **)
    ENERGY := RMAX;
  DLY := ENERGY/RMAX; (** REST BASED UPON ENERGY **)
  IF DLY < -ENERGY THEN
    DLY := -ENERGY;
  DELAY(DLY); (** REST **)
UNTIL SCREEN.DONE(X,Y,ENERGY); (** DECIDE WHETHER STILL "ALIVE" **)
END; (** PROCESS RABBIT **)

```

PROCEDURE PROCREATE: (** CREATE RABBITS AND WOLVES **)

```

VAR
  X : INTEGER; (** CREATOR'S CURRENT POSITION **)
  Y : INTEGER;
  ENERGY : INTEGER; (** CREATED'S INITIAL ENERGY **)
  DIR : (UP,DOWN,LEFT,RIGHT); (** CURRENT DIRECTION **)
  DIRS : INTEGER; (** DIRECTION COUNTER **)
BEGIN
  FOR Y := 1 TO YMAX DO (** INITIALIZE SCREEN **)
  BEGIN
    SCREEN ASSIGN(XWOLF,Y,MINIT); (** A COLUMN OF WOLVES **)
    WOLF(XWOLF,Y,MINIT);
    SCREEN ASSIGN(XRABBIT,Y,RINIT); (** A COLUMN OF RABBITS **)
    RABBIT(XRABBIT,Y,RINIT);
  END;
  DIR := UP; (** CREATION DIRECTION **)
  REPEAT (** CREATE OFFSPRING AS REQUIRED **)
  CREATOR STARTUP(X,Y,ENERGY); (** OBTAIN OPERATION REQUEST **)
  IF ENERGY > 0 THEN (** REQUEST FOR CREATION **)
  BEGIN
    DIRS := 4; (** TRY ALL FOUR DIRECTIONS **)
    REPEAT
      IF DIR=RIGHT THEN (** CONSIDER NEXT DIRECTION **)
        DIR := UP;
      ELSE
        DIR := DIR(DIR);
      CASE DIR OF
        UP : IF Y=1 THEN
          IF SCREEN.CHANGE(X,Y,ENERGY) THEN
            BEGIN
              DIRS := 0; (** SET FLAG **)
              Y := Y+1; (** UPDATE DIRECTION **)
            END;
          DOWN : IF Y=2-YMAX THEN
            IF SCREEN.CHANGE(X,Y+1,ENERGY) THEN
              BEGIN
                DIRS := 0; (** SET FLAG **)
                Y := Y+1; (** UPDATE DIRECTION **)
              END;
          LEFT : IF X=1 THEN
            IF SCREEN.CHANGE(X-1,Y,ENERGY) THEN
              BEGIN
                DIRS := 0; (** SET FLAG **)
                X := X-1; (** UPDATE DIRECTION **)
              END;
          RIGHT : IF X=XMAX THEN
            IF SCREEN.CHANGE(X+1,Y,ENERGY) THEN
              BEGIN
                DIRS := 0; (** SET FLAG **)
                X := X+1; (** UPDATE DIRECTION **)
              END;
        END;
      DIRS := DIRS+1;
    UNTIL DIRS=0;
    IF DIRS=0 THEN (** IF 0, CANNOT DO CREATION **)
      IF ENERGY > 0 THEN (** CREATE NEW RABBIT **)
        RABBIT(X,Y,ENERGY);
      ELSE (** CREATE NEW WOLF **)
        WOLF(X,Y,ENERGY);
    END;
  UNTIL ENERGY < 0; (** SHUTDOWN PROVIDES SPECIAL CODE FOR TERMINATION **)
  SHUTDOWN(0); (** SHUTDOWN HANDLES EXTRANEOUS CREATION REQUESTS **)
END; (** PROCEDURE PROCREATE **)

```

PROCESS SHUTDOWN: (** HANDLE TERMINATION **)

```

VAR
  CH : CHAR; (** INPUT CHARACTER **)
  X : INTEGER; (** ABSORB EXCESS ATTEMPTS TO PROCREATE **)
  Y : INTEGER;
  ENERGY : INTEGER;
BEGIN
  CRTIRUP;POP(CH); (** WAIT FOR TERMINAL INPUT **)
  SCREEN KILL; (** NOTIFY SCREEN MONITOR **)
  CREATOR.CREATE(0,0,0); (** TERMINATE PROCEATION **)
  SHUTUP WAIT;
  REPEAT (** HANDLE ANY EXTRANEOUS ATTEMPTS AT CREATION **)
  CREATOR.STARTUP(X,Y,ENERGY);
  UNTIL ENERGY < 0;
END; (** PROCESS SHUTDOWN **)

```

PROCEDURE PARAMS: (** PROMPT USER FOR INPUT **)

```

TYPE
  STRINGG = PACKED ARRAY [1..64] OF CHAR;
PROCEDURE PARAMIN(X,Y:INTEGER; MSG:STRINGG; VAR VAL:INTEGER);
VAR
  I : INTEGER;
  CH : CHAR;
BEGIN
  CRTIRUP;PUSH(DIR,ESC); (** VT52 JUMP **)
  CRTIRUP;PUSH(Y);
  CRTIRUP;PUSH(CHR(Y+1));
  CRTIRUP;PUSH(CHR(X+1));
  FOR I := 1 TO 64 DO (** PROMPT MESSAGE **)
    CRTIRUP;PUSH(MSG[I]);
  VAL := 0;
  REPEAT (** READ AND DECODE VALUE **)
    CRTIRUP;POP(CH);
    IF CH = 0 AND (CH = 0) THEN
      BEGIN
        VAL := VAL+(ORD(CH)-ORD('0'));
        CRTIRUP;PUSH(CH);
      END;
    ELSE
      IF CH = CHR(0) THEN (** UNRECOGNIZED **)
        CRTIRUP;PUSH(CHR(0));
      UNTIL VAL < 0 OR VAL > 65535;
  END; (** PROCEDURE PARAMIN **)

```

```

BEGIN
  PARAMIN(XMAX+1,MINIT,MINIT); (** INITIAL ENERGY FOR WOLVES **)
  PARAMIN(XMAX+1,RINIT,RINIT); (** INITIAL ENERGY FOR RABBITS **)
  PARAMIN(XMAX+1,WYR,WYR); (** NORMAL DEFTIME FOR WOLVES **)
  PARAMIN(XMAX+1,EYR,EYR); (** NORMAL DEFTIME FOR RABBITS **)
  WANNAL := WINIT DIV WYR; (** WOLF VALUES ARE POSITIVE **)
  WEFTILE := WANNAL;
  WRABW := RINIT;
  WMAX := RMINIT;
  RINIT := RINIT; (** RABBIT VALUES ARE NEGATIVE **)
  RANNAL := RINIT DIV RYR;
  REFTILE := RANNAL;
  RRABW := RMINIT;
  RMAX := RMINIT;
END; (** PROCEDURE PARAMS **)
BEGIN (** PROGRAM ISLAND **)
  NEW(CRTIRUP); (** CREATE INPUT BUFFER **)
  DIV(IN(CRTIRUP)); (** STARTUP INPUT PROCESS **)
  NEW(CRTORUP); (** CREATE OUTPUT BUFFER **)
  DIV(OUT(CRTORUP)); (** STARTUP OUTPUT PROCESS **)
  REPEAT
    CRTORUP;PUSH(CHR(ESC)); (** VT52 HOME OPERATION **)
    CRTORUP;PUSH(CH);
    CRTORUP;PUSH(CHR(ESC)); (** VT52 CLEAR SCREEN OPERATION **)
    CRTORUP;PUSH(' ');
  PARAMS; (** ASK USER FOR PARAMETERS **)
  SCREEN SETUP; (** INITIALIZE INTERNAL SCREEN **)
  SHUTDOWN; (** SETUP FOR EVENTUAL TERMINATION **)
  PROCREATE; (** RETURNS WHEN ALL TASKS COMPLETED **)
  CREATOR.CREATE(0,0,0); (** TERMINATE SHUTDOWN **)
  UNTIL FALSE;
END; (** PROGRAM ISLAND **)

```

PUG

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 für 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,P1;
  VAR a,b,c: INTEGER;

  PROCEDURE P1;
  BEGIN
    a := a + 1;
    x := a;
  END P1;

END Mod;

. . . . .

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

(* no module here *)
  a,b,c: INTEGER;

  PROCEDURE P1;
  BEGIN
    a := a + 1;
    x := a;
  END; (* P1 *)

. . . . .

END Outside;

END; (* Outside *)
```

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 as 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:

```

MODULE MainProgram;
. . . . .
MODULE RandomNumbers;
IMPORT TimeOfDay;
EXPORT Random;
CONST Modulus = 2345;
      Increment = 7227;
VAR Seed : INTEGER;

PROCEDURE Random() : INTEGER;
BEGIN
  Seed := (Seed+Increment)
        MOD Modulus;
RETURN Seed;
END Random;

BEGIN (* RandomNumber *)
  Seed := TimeOfDay;
END RandomNumber;

. . . . .
BEGIN (* MainProgram *)
. . . . .
WriteInt(Random(), 7);

END MainProgram.

PROGRAM MainProgram;
VAR Seed : INTEGER;
. . . . .
FUNCTION Random : INTEGER;
CONST Modulus = 2345;
      Increment = 7227;
BEGIN
  Seed := (Seed+Increment)
        MOD Modulus;
  Random := Seed;
END; (* Random *)

. . . . .
BEGIN (* MainProgram *)
  Seed := TimeOfDay;
. . . . .
WriteInt(Random, 7);

END. (* MainProgram *)

```

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 RandomNumber 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 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;      10: Remainder := Alpha MOD Beta;
WHILE Remainder <> 0 DO           IF Remainder = 0 THEN
  BEGIN                           GOTO :20;
  Alpha := Beta;                  Alpha := Beta;
  Beta := Remainder;              Beta := Remainder;
  Remainder := Alpha MOD Beta     GOTO 10;
  END;                             20: . . . . .

```

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;
  Beta := Remainder
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 statement which terminates the range of the LOOP.

Additional examples of unconditional transfers supported by Modula-2 include the RETURN statement 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 procedures (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 operations 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. Modula-2 programs are constructed from two kinds of compilation 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. Definition modules contain declarations of the objects which are exported to other compilation units. Implementation 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 library module into separate definition and implementation 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 compilation units. Defining and maintaining consistent interfaces is of critical importance in ensuring error-free

communication between subsystems. During the design 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 subsystem 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 enforced 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 kinds of modules:

1. Low-level system modules which provide access to local system resources;
2. Standard utility modules which provide a consistent 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 containing compiled forms of the library module's compilation units. The library is accessed by both the compiler and the program loader — the former reads the compiled definition modules while compiling and the latter loads the compiled implementation modules when executing 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. Therefore, the definition module must be compiled first. Once an implementation module has been compiled, its object 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 imports 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 procedures for I/O, memory allocation, or process scheduling. 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 modules, Modula-2 programs become portable across all implementations.

The advantages of expressing commonly-used routines as library modules (rather than part of the language) include a smaller compiler, smaller run-time system, and the ability to define alternative facilities when the standard facilities prove insufficient. Disadvantages 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
2. Niklaus Wirth, *MODULA-2*, ETH Institut fur Informatik Report No. 36, reprinted by Volition Systems, Del Mar, CA, 1980
3. Rich Gleaves, *Modula II User's Manual*, Volition Systems, 1982
4. Roger Sumner and Rich Gleaves, *Modula-2 — A Solution to Pascal's Problems*, Volition Systems, 1982

PUG

AVAILABLE ON MICROFICHE

DIRECT INQUIRIES TO:

MICRO PHOTO DIVISION

 **BELL & HOWELL**

OLD MANSFIELD ROAD
WOOSTER OH 44691
Contact Christine Ellis
Call toll-free (800) 321-9881
In Ohio, call (216) 264-6666 collect



Data Structures Using Pascal
by A.M. Tenenbaum and M.J. Augenstein
Prentice-Hall, 1981

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 al-

gorithmic 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 Pascal/Z[®] compiler marketed by Ithaca InterSystems, Inc.; Laurie Hanselman Moskow, formerly Software Products 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 Building Blocks, Inc. is a two-pass, locally optimizing Pascal compiler for the IBM Personal Computer. The initial release will run under PC-DOS[®]; and a CP/M-86[®] version is planned for the near future. Based on the Pascal/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, external routines, include files, symbolic I/O of enumeration types, an ELSE clause for the CASE statement, overlays and chaining.

The compiler package includes a sophisticated interactive Pascal debugger, written in SBB Pascal, designed to aid in isolating and correcting faults in Pascal programs. Features of the debugger include the abilities to set and display both absolute and conditional breakpoints; set watches on variables, procedures or functions; display and modify both global and local variables; display the procedure/function stack, current statement and module numbers, current run-time requirements, and the last ten statements executed; trace through a program by statement number and procedure/function entry/exit; and more.

Also included in the package is a screen editor, provided in SBB Pascal source. The editor's capabilities include: insertion and deletion of lines and characters, finding and/or replacing of strings, copying lines of text, autoindent for entering structured programs, and many other features. The editor makes use of the function keys on the IBM PC to make editing as easy and efficient as possible. The editor is provided in source as an example of the advantages of programming in SBB Pascal. The library routines are also provided in 8086 assembly language source, and many other example programs are included as well.

Software Building Blocks, Inc. intends to release the PC-DOS version of the compiler in June. For more information, contact Laurie Moskow, Software Building Blocks, Inc., P.O. Box 119, Ithaca, New York, 14851-0119, (607) 272-2807.

® CP/M-86 is a trademark of Digital Research, Inc.

- ® IBM and IBM Personal Computer are registered trademarks of International Business Machines Corporation
- ® 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 trademarks of Software Building Blocks, Inc.

SAGE OPENS BOSTON DIVISION

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

The purpose of the new facility is to provide 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 company'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. McDermott, 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 accommodates up to 6 simultaneous users, surpasses the considerable capabilities of the Sage II introduced in March, 1982.

Both machines are based on the 8 MHz 68000 processor, 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 mini-computers at a mid-range to high-end business micro price.

The Sage IV comes standard with 128K of main

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 LANGUAGE. 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 process control; console and file I/O, including random access files and disk directory operations; and storage management. The standard software modules also include utility routines for format conversion, strings, 19 digit BCD arithmetic, and other facilities.

The manual is divided into six sections. The Modula-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 utility library, a system document that describes the implementation of Modula-2 for UCSD Pascal[®] and a machine-specific implementation guide which includes information on machine specific library modules, interrupt handling, and machine-level data representation.

The Modula-2 User's Manual, including Wirth's Modula-2 report, is immediately available from Volition 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 software 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. 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 systems, 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

USUS FALL MEETING SET FOR WASHINGTON, D.C.

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.

UCSD PASCAL USERS FORM TEXT EDITOR INTEREST GROUP

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, non-profit 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.

PASCAL USER'S SOCIETY FORMS MODULA-2 GROUP

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 IIe including extended memory support for the 128K IIe. 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, Corvus 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-

retary, 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 members 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 Pascal¹ 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 III 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-NIX³ 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 III are trademarks of Apple Computer, Inc.

³ UNIX is a trademark of Bell Laboratories.

PUG

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. *)

MC 6809 Processor
Running Moos, 05-9, 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 *)

Warshalls 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.

Validation Suite Reports Validation Suite Reports Validation Suite Reports Validation Suite Reports Validation

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 Jenson/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.

Test 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 longinteger 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).

a 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.

PUG

Pascal News

2903 Huntington Road
Cleveland, Ohio 44120

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:

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.

Pascal News
2903 Huntington Road
Cleveland, Ohio 44120

Back issues are requested and sent in sets

OUT OF PRINT

- \$25 set 1 Issues 9 . . . 12 (September 1977 — June 1978)
- \$25 set 2 Issues 13 . . . 16 (December 1978 — October 1979)
- \$25 set 3 Issues 17 . . . 20 (March 1980 — December 1980)
- \$25 set 4 Issues 21 . . . 23 (April 1981 [mailed January 1982] —
September 1981 [mailed March 1982])

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:
 ASCH 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. Compu-Serve 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 aides 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):

<input type="checkbox"/> Advanced System Editor SIG	<input type="checkbox"/> Meetings Committee
<input type="checkbox"/> Apple SIG	<input type="checkbox"/> Modula-2 SIG
<input type="checkbox"/> Application Developer's SIG	<input type="checkbox"/> NEC Advanced PC SIG
<input type="checkbox"/> Communications SIG	<input type="checkbox"/> Publications Committee
<input type="checkbox"/> DEC SIG	<input type="checkbox"/> Sage SIG
<input type="checkbox"/> File Access SIG	<input type="checkbox"/> Software Exchange Library
<input type="checkbox"/> Graphics SIG	<input type="checkbox"/> Technical Issues Committee
<input type="checkbox"/> IBM Display Writer SIG	<input type="checkbox"/> Texas Instruments SIG
<input type="checkbox"/> IBM PC SIG	<input type="checkbox"/> 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.

Pascal News

2903 Huntington Road
Cleveland, Ohio 44120

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:

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.

Pascal News
2903 Huntington Road
Cleveland, Ohio 44120

Back issues are requested and sent in sets

OUT OF PRINT

\$25 set 1 Issues 9 . . . 12 (September 1977 — June 1978)

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

\$25 set 3 Issues 17 . . . 20 (March 1980 — December 1980)

\$25 set 4 Issues 21 . . . 23 (April 1981 [mailed January 1982] —
September 1981 [mailed March 1982])

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 _____

Pascal News
2903 Huntington Road
Cleveland, Ohio 44120

Back issues are requested and sent in sets

~~Set 1: Issues 9-12 (Sept 1977 - June 1978)~~ **OUT OF PRINT**

\$25 set 1 Issues 9 . . . 12 (September 1977 — June 1978)

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

\$25 set 3 Issues 17 . . . 20 (March 1980 — December 1980)

\$25 set 4 Issues 21 . . . 23 (April 1981 [mailed January 1982] —
September 1981 [mailed March 1982])

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

Pascal News

2903 Huntington Road
Cleveland, Ohio 44120

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:

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.

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:

Pascal — User Manual and Report (Second, study edition)
by Kathleen Jensen and Niklaus Wirth.
Springer-Verlag Publishers: New York, Heidelberg, Berlin
1978 (corrected printing), 167 pages, paperback, \$7.90.

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.

Return to:

Pascal News

2903 Huntington Rd. • Cleveland, Ohio 44120

Return postage guaranteed Address Correction requested

BULK RATE
U.S. POSTAGE
PAID
WILLOUGHBY, OHIO
Permit No. 58

This is your last issue if you have not renewed for 1983!

FRAS

Andy Mickel
copy two
100 SE Ave
Minneapolis MN